Contents

1. 2014

1.1 February

Introduction (2014-02-08 13:05)

Why Learn C++?

- Very powerful and respected prog. Lang.
- Highly respected in Academia
- B. Tech./M.Tech. Project work
- Job Written Test + Interview
- Research: Programming work can be done in C++

Only course in C++

- This may be the only course in C++ done by you.
- So learn it well
- Benefits, later on in life, can be enormous
- Excellent Lab facilities
- Use extra Lab time whenever possible
- But do not neglect other courses

Course Book

- C++ Primer, Fourth Edition
- Authors:
  - Stanley B. Lippman
  - Josee Lajoie
  - Barbara E. Moo
- Publisher: Pearson Education
- ISBN: 81-317-1077-7
- Price
  - List Price: Rs. 475
  - Discount: Usually 10 to 20 %

Course Book - 2

- Availability of hard copy enables:
  - Easy lookup and study
  - Faster and better learning
- At least one copy for two students is essential
- Will be useful after course too
- Procure book quickly
- Book covers both C and C++
- Reading chapters of book will be very useful

First C++ Program

```
#include <iostream> //Not iostream.h
int main()
{
    std::cout << “Hello world\n”;
}
```

First C++ Program - 2

- C++ is a superset of C
- Standard header files do not have .h typically
- The iostream header includes information about stream objects used for input and output.
- All such stream objects are said to be part of the iostream library
- Instead of printf we use std::cout

First C++ Program - 3

- std is a namespace
- cout is an object which writes (outputs) to a stream (the terminal usually)
- :: is required between namespace and object
- << is the way to direct data into cout object
- << is also called output operator

First C++ Program - 4

- Source code file extension should be .cpp
- Save file as firstpgm.cpp
- Compile using c++ or g++
- Executable file is named as a.out

Second C++ Program

- Book code is available as a zip file
  - Windows: http://www.informit.com/content/images/0201721481/sourcecode/MS_files.zip
  - Linux: http://www.informit.com/content/images/0201721481/sourcecode/gnu_files.tar.gz
- Copy it to your system (on Linux) and unzip it
- Chapter 1, add.cc
  - .cc is also used as extension for C++ source files but you should only use .cpp while you are doing this course.

Second C++ Program - 2

- << output operator can be used multiple times in one statement
- std::endl is another way of inserting a newline (\n)
- Defining int variables is same as C

Second C++ Program - 3

- std::cin is an input stream object which usually reads from keyboard
- >> is the input operator which reads input and puts the input into variables on right hand side of operator

Read Course Book

- Course book (C++ Primer 4/e):
- Gives a lot of additional information far beyond what we have time to cover in class
- Is somewhat precise and so may take getting used to

Read Course Book - 2

- But it is very good and very accurate
- Has simple exercises
- Has nice layout with certain paragraphs marked specially as Notes or Beware (warnings)
- Reading and understanding it thoroughly will make you a thorough C++ Guru

Hands On (Finally :-))

- Start your system
- Choose Red Hat Linux as OS (instead of default Windows)
- Log on with userid: — and password: —
- Wait for the GUI (Graphical User Interface) to come up properly
- Open a terminal window

First Assignment

- Write the Hello World program using vi
- Source file name must end in .cpp
- Compile it using c++ or g++
- Executable file will be a.out (like what cc does)
- Run ./a.out

Second Assignment

- First copy the book code zip file onto your system
- Unzip the file
- Now read 1/add.cc
- Then write a program which takes in four integral numbers and gives the average of them as output.

Introduction - 2

End of Input

- See 1/mysum.cc (Section 1.44 in book)
- Use Ctrl-D to signify End of data from Keyboard
- while (std::cin >> value)
  - Operation is executed
  - std::cin is evaluated (true/false)
  - If operation succeeded cin is true
  - If operation failed (invalid input, end of data) cin becomes invalid and returns false

Sales _item class

- Chapter 1 of Course Book introduces this class
- I prefer to skip classes (C++ classes :-)) till later

Chapter 2

- Variables and Basic Types
- Built-In types: int, char, long, float, double etc.
- Literals: 128, 3.14f, ‘a’, “Rama”
- Variables: int a = 2;
- variable vs. object: For now we can use the word interchangeably

const object/variable

- const int bufsize = 512;
- Value cannot be changed

Temporarily skipping

- References (Sec. 2.5)
- Typedefs (Sec 2.6)
- Enums (Sec 2.7)
- Will cover them later

C++ Classes

- Covered in Section 1.5, 1.6, 2.8
- We will do parts of section 2.8 first

C++ class

- C++ allows programmers (us) to define our own types (in addition to built-in types like int)
- These types are known as classes
- Variables can be defined for these classes just like variables for built-in data types like int

Variable/Object

- int v1;
  - v1 is a variable of type int
  - v1 is an object of type int

Sales _item class

- Sales _item book;
  - We assume Sales _item is defined elsewhere and accessible to us using header file Sales _item.h
  - See 1/item _io.cc
  - book is a variable of type Sales _item
  - book is an object of type Sales _item
  - book is an object of class Sales _item (common usage)

Sales _item

- Multiple objects can be defined of Sales _item
- Sales _item item1, item2, item3;

- 3 Sales _item objects have been created
- Each object will have its own data
- Similar to multiple variables/objects of built-in types.

struct vs class

- C++ class has all functionality of ‘C’ struct
- And has more functionality than struct
- Note that struct is supported in C++ also and further note that C++ struct is more powerful that C struct.

Type Safety

- C and C++ are type-safe languages
- Compiler knows data type of variables/objects and disallows inappropriate operations
  - E.g. Passing a character pointer where a float argument is expected
- Type-safe languages force some amount of discipline on the programmer (otherwise – lots of compiler error and warning messages).

Object jargon

- Sales _item item;
- Based on the above code the following jargon can be used:
  - item is an object of type Sales _item
  - item is a Sales _item object
  - item is a Sales _item
- This is similar to jargon used for variables/objects of built-in types

Introduction - 3

References

- Section 2.5 in course book
- A reference serves as an alternative name for an object
- In real-world programs, references are primarily used as parameters to functions
- Reference is different from pointer
- Defined by preceding a variable name by the & symbol

Reference Example

- int ival = 1024;
- int &refval = ival;
- refval is a reference to ival
- refval += 2; // adds 2 to ival
- int i1 = refval; // assigns value of ival to i1

Reference examples

- A reference MUST be initialized.
- int &refval2; //error

- A reference must be initialized using an object of the same type as the reference
- int &refval3 = 10; // error – initializer must be an object

Reference vs pointer

- Pointer uses indirection syntax
- int *iptr = &ival;
- *iptr = 5;
- Note that & symbol is used both for references and addresses; From context we know whether it is a reference or an address.

Reference vs pointer - 2

- Pointer is a separate variable whose value can be independently changed. Reference is only an alias to an object and does not have a separate existence
- iptr = null;
- Pointer can be changed to point to another object
- Reference can never be changed to become an alias for another object

Const references

- A const reference is a reference that refers to a const object (or literal value)
- const int ival = 1024;
- const int &refval = ival; //ok
- int &ref2 = ival; // error: nonconst reference to a const object
- const int &r = 42; //ok as it is a const reference

Exercise

- Solve Exercises section 2.5 from course book. Write observations/answers in a text file
- Write a program that:
  - Defines double variables
  - Defines non-const and const references to them
  - Defines pointers to them
  - Then write code which uses the references and pointers – objective of the code is to promote your understanding of references.

Typedef

- A typedef defines a synonym for a type
- typedef double wages;
- wages hourly, weekly;
- typedef struct { … } abc;
- abc var1;

Enumerations

- File open mode typically may be one of three values: input, output or append
- To keep track of openmode we could use three integer constants:
  - const int input = 0;
  - const int output = 1;
  - const int append = 2;
- int fileopenmode = input; //variable to keep track of mode

Enum

- Better way to handle situation is to use enum
- enum open _modes {input, output, append };
- Defines an enumeration type called open _modes which can hold only one of the three listed values.
- open _modes fileopenmode = input;
- Fileopenmode variable is of type open _modes and can be assigned one of its enumerations ONLY.

Enum - 2

- Enum superior to plain const:
  - validation is automatically performed
  - set of enumerations are defined at one place.
- enum variables are assigned integral values with first enumeration being 0 by default and the others increasing by 1
- See enumtest.cpp

Exercise

- Write an enum type for weekday – Sunday, Monday …
- Define two variables of that enum type
- Assign some values for both variables in the program.
- Print out the day of the week for both variables (string should be printed and not 0 or 1 …)
- Compare the two variables and print whether they are equal or not.

—–

Note: The course book is C++ Primer, 4th Edition by Lippman, Lajoie and Moo. References above to source files, and use of code within those files, are of example code given in and/or along with the book. As this post is freely accessible on the Internet, not-for-profit, and for educational purposes, based on the permission related statements in the source code, I have considered that permission has been granted to use them here.

Introduction to C++ Classes (2014-02-11 12:06)

Note: Some of the slides-contents below are repeated from Introduction to C++

C++ Classes

- Covered in Section 1.5, 1.6, 2.8
- We will do parts of section 2.8 first

C++ class

- C++ allows programmers (us) to define our own types (in addition to built-in types like int)
- These types are known as classes
- Variables can be defined for these classes just like variables for built-in data types like int

Variable/Object

- int v1;
  - v1 is a variable of type int
  - v1 is an object of type int

Sales _item class

- Provides the following operations:
- >> input operator to read a Sales _item object
- << output operator to write a Sales _item object
- + addition operator to add two Sales _item objects
- = assignment operator to assign one Sales _item object to another
- same _isbn() member function

Sales _item class - 2

- These operations/functions of Sales _item are defined elsewhere and accessible to us using header file Sales _item.h
- Sales _item book;
  - See 1/item _io.cc
  - book is a variable of type Sales _item
  - book is an object of type Sales _item
  - book is an object of class Sales _item (common usage)

Sales _item

- Multiple objects can be defined of Sales _item
- Sales _item item1, item2, item3;
  - 3 Sales _item objects have been created
  - Each object will have its own data
  - Similar to multiple variables/objects of built-in types.

struct vs class

- C++ class has all functionality of ‘C’ struct
- And has more functionality than struct
- Note that struct is supported in C++ also and further note that C++ struct is more powerful that C struct.

Type Safety

- C and C++ are type-safe languages
- Compiler knows data type of variables/objects and disallows inappropriate operations
  - E.g. Passing a character pointer where a float argument is expected
- Type-safe languages force some amount of discipline on the programmer (otherwise – lots of compiler error and warning messages).

Object jargon

- Sales _item item;
- Based on the above code the following jargon can be used:
  - item is an object of type Sales _item
  - item is a Sales _item object
  - item is a Sales _item
- This is similar to jargon used for variables/objects of built-in types

Member Functions

- See add _item2.cc in folder 1
- A member function is a function defined (provided) by a class
- Member functions a.k.a. Methods
- Dot operator used to invoke member function in the context of an object
- item1.same _isbn(item2)
- same _isbn() method of item1 is called.

Addition operator

- Adding two Sales _item objects creates a new Sales _item object whose:
  - ISBN is that of the operands
  - Number sold and revenue reflects the sum of the corresponding values of the operands
  - For input: [ISBN, no. sold, price of each]
    - 0-201-78345 3 20.00
    - 0-201-78345 2 25.00
  - Output is: [ISBN, no. sold, total revenue, average price]
    - 0-201-78345 5 110 22

Running add _item2.exe

- Build programs in folder 1 by using command make
- Run add _item2.exe by:
  - $ ./add _item2.exe
  - Will prompt you for Sales _item data (ISBN, no. sold and price of each)
  - Will print out sum

Running add _item2.exe - 2

- You can run programs which use standard input (cin) against a data file by redirection
- Data file for add _item2.exe is data/add _item2
- ./add _item2.exe < data/add _item2
  - Runs add _item2.exe with input taken from file data/add _item2

Redirecting output

- ./add _item2.exe < data/add _item2 > data/xyzsum
  - Also redirects output to file data/xyzsum
  - Be careful to choose file names different from that already in data
- Can copy folder 1 to a working directory and safely experiment
  - cp –R src-folder dest-folder
    - Command for recursive (folders and contained sub-folders) copy

Linux User Level Knowledge

- What are the commonly used Linux text based commands?
- How to become a power user?
- What are the commonly used GUI programs on Linux?
- Check out the Internet for the above

Exercises

- Copy folder 1 to a working directory
- ‘make’ the executables in that directory
- Run program add _item2 with keyboard data and file data
- Save output of program into a data file and view it.

- Read add _item2.cc and understand it
- Go through Linux reference guides for newbies

Think About - Exercise

- Exercise 1.24 (Section 1.5.2)
- Program reads several transactions
- For each new transaction determine if it is the same ISBN as previous transaction
- Keep a count of how many transactions there are for each ISBN
- Test program with test data (file)
- Test data should have multiple transactions with transactions for same ISBN grouped together

Exercise 1.24 & Assignment

- See avg _price.cc
- Assignment: Modify avg _price.cc to print individual transactions followed by total sales info (total sales info is as is already given by avg _price.cc)
- Test your solution with existing data (data/book _sales).
- If there are bugs in your program mention it as comments in your source code.

Read Course Book

- Chapters 1 and 2 of course book have been covered (barring few exceptions)
- Please read chapters 1 and 2 of course book

—–

Note: The course book is C++ Primer, 4th Edition by Lippman, Lajoie and Moo. References above to source files, and use of code within those files, are of example code given in and/or along with the book. As this post is freely accessible on the Internet, not-for-profit, and for educational purposes, based on the permission related statements in the source code, I have considered that permission has been granted to use them here.

1.2 March

Status History of Putting Up C++ Programming Course Content (2014-03-18 06:49)

The sections below deal with the status of putting up course content on this blog over the past few months.

Status around August 2013

I have written to Pearson Education India seeking permission from them for putting up those of the slides that have examples and maybe some other material from the course book. I hope to receive their permission as the slides will be promoting their book. Students interested in seriously using the slides and other course content here would need to buy/refer the book for details. Such a purchase would usually stand students in good stead even after going through this course as the book can be a companion or reference for future work involving C++ like an M.Tech. (CS) project or even initial period of software industry programming career.

Status update on 10th February 2014

In August 2013 I had interacted with the appropriate person of Pearson Education India, Rights and Permissions department (http://www.pearsoned.co.in/web/Rights.aspx), over email and telephone which resulted in me sending them (by email) a zip file on 21st August 2013 with my C++ programming course slides. I have copy-pasted the main body of the mail below:

Further to our telephone discussion today afternoon please find attached a zip file which has all the C++ course slides, that I intend to put up on my site here: http://raviiyerteaches.wordpress.com/2013/08/15/cpp-programming/. As the book, C++ Primer, 4th edition by authors Lippman, Lajoie and Moo, http://www.pearsoned.co.in/prc/book/stanley-b-lippman-c-primer-4e–4/97881317 10777, is the course book for the C++ course mentioned above, many of the slides refer to examples and sections of the book.

I seek permission from Pearson Education for putting up these slides containing examples from the above mentioned book, on the Internet at the site/url mentioned above.

I would be happy to include one slide at the end of each of the above slide files thanking Pearson Education for permission provided and having specific attribution to the above mentioned book.

Please note that putting up the above slides on the Internet may lead some people to buy the above mentioned book.

Further please note that I am aware of a later edition (5th edition) of the book being available in the market. I may, in future, consider upgrading the course content to use the later edition. At that time I will seek fresh permission from Pearson Education to put the modified/new slides on the Internet.

— end main body of mail to Pearson Education India —

However, I have so far (10th February 2014), not received a response from Pearson Education India for the above mentioned mail.

Today I got a pleasant surprise regarding grant of permission for educational purposes when I looked at an example source code file of chapter 1 (folder named 1) - mysum.cpp. The appropriate part of the commented header section of this file is given below:

Permission is granted for this code to be used for educational purposes in
association with the book, given proper citation if and when posted or
reproduced.Any commercial use of this code requires the explicit written
permission of the publisher, Addison-Wesley Professional, a division of
Pearson Education, Inc. ...

—- end extract from header comments of source file –

As this blog/website of mine is a free teaching service to students on the Internet, and not-for-profit, I can consider that permission has been granted for me to use code from the above file on this blog. Further, I can use code from all other example code files of the book which carry a similar permission grant statement (which I presume they have, and will confirm before use) on this blog/website. I think that solves my problem. Of course, I will give proper citation on each blog post/web page that uses the book code.

Status update on 17th March 2014

Now I have put up all the required content, suitably modified slightly, on this blog from the C++ course content I had prepared and used to teach the course in the deemed university in Andhra Pradesh, India.

Advanced Unix Programming **(2014-03-18 09:50)**

Last updated on 24th March 2014
This course may have been last taught by me in 2008 in a deemed university in Andhra Pradesh, India. I

may have taught this course for four or five batches/years. The class size was typically around 14 to 18 students.

This is a Lab course I taught for I. M.Tech. (Comp. Sc.). It teaches usage of Unix system calls. For example, writing a shell program is one of the assignments in the course where the student will have to use fork, exec and waitpid system calls. However the course does not get into implementation of these system calls itself (kernel programming).

Knowledge of Unix system calls (also referred to as functions) programming should enable the student to easily learn how to use corresponding system calls in other operating systems like Windows. For example once the student completes this course successfully he/she should be able to easily pick up how to programmatically create processes in Windows and how to programmatically navigate through the Windows file system.

Prerequisites for the Course
C Programming including debugging skills and User level knowledge of Unix (Unix commands like ls, chmod, mkdir, kill etc.)

Course Book
Advanced Programming in the Unix Environment, 2nd edition by W. Richard Stevens and Stephen A. Rago. Here's the book support site: http://www.apuebook.com/apue2e.html. A 3rd edition is now available (dated 2013) but I have not read the book and so do not know its suitability for this course. I believe that the 2nd edition pdf version (and the 3rd edition pdf version - http://it-ebooks.info/book/3040/) is available for free download (legally).

Ideal Coverage (subject to time limitations)

1. Building large programs with multiple source files, related compiler options, library archive files (.a files), using make to build programs.
2. Quick overview of Unix system architecture, error handling, system calls and library functions.
3. Self Study of Unix Standardization and implementations: IEEE POSIX, Single Unix Specification, POSIX limits, sysconf, pathconf.
4. File I/O: open, lseek, read, write, dup, fcntl functions; atomic operations
5. Files & Directories: stat, umask, chmod, chown, link, unlink symlink, readlink, mkdir, rmdir chdir, getcwd functions.
6. Process Environment: Process termination, environment list, environment variables
7. Process Control: fork, exit, wait, waitpid, exec, system; race conditions, synchronization
8. Self Study of Process Relationships: Process groups, sessions, controlling terminal, job control, shell execution of programs.
9. Signals: Signal concepts, signal function, unreliable signals, interrupted system calls, reentrant functions, kill, raise, alarm and pause functions; signal sets, sigprocmask, sigpending, sigaction, sigsuspend, abort, system functions.
10. Self Study of Daemon processes: Daemon characteristics, coding rules, error logging

11. Advanced I/O: select function and quick overview of record locking
12. Interprocess Communication: pipe, popen, pclose, fifo functions, message queues, semaphores, shared memory related functions.

For all the topics excepting Self Study topics programming assignments are given to the students.

Note that in some years, due to paucity of time, topics 7 to 9 were covered in a rushed manner and topics 10 to 12 were skipped. For the 2004 batch there was enough time to cover all the topics mentioned above (including assignments).

Also note that MultiThreaded programming is not covered in this course due to lack of time. However another course, Network programming, covers Multi Threaded programming.

Index of Links to Chapter Wise Course Pages

The following two links have slides and some examples from US professors on the same subject and based partly on the same course book:
1) http://www.cs.stevens.edu/ jschauma/810/
2) http://www.cs.fsu.edu/ asriniva/courses/aup02/lectures.html

The above links were obtained via Google search and so presumed to be freely accessible to any-body on the net.

For those students who are not familiar with make, slides and examples of lectures 1 & 2 of the second link above (cs.fsu.edu) will be useful.

Intro.ppt

Please note that only three of the chapter links below have Powerpoint slides links (some of which have very few slides - topics only type). If I recall correctly I used slides from the above-mentioned US university links for some of the chapters - why reinvent the wheel? However the chapter links below clearly specify the sections covered/to be read as reading assignments and the assignments, most, if not all, of which were created by me specially for this course. So I think they give a good framework for students to learn Advanced Unix Programming in a semester in college/university environments.

Further, please note that, as a first step, I have focused on putting up the course content used by me to teach this course in the deemed university in Andhra Pradesh, India, suitably modified, on this blog. As part of the minor modifications for putting it up on this publicly accessible blog, some errors may have crept in. I have not checked all the modifications for accuracy (due to other demands on my time). If this course (on this blog) does get utilized by students then the errors in the modified part will come to light and will get fixed by me (or others). I think that is a better way of investing my (and others) time for fixing any errors in this course (on this blog).

Chapter 1 - Unix System Overview

Chapter 2 - Unix Standardization and Implementations (Reading Assignments only)

Chapter 3 - File I/O

Chapter 4 - Files and Directories

Chapter 5 - Standard I/O Library (Reading Assignments only)

Chapter 6 - System Data Files and Information (Reading Assignments only)

Chapter 7 - Process Environment

Chapter 8 - Process Control

Chapter 9 - Process Relationships

Chapter 10 - Signals

Chapter 14 - Advanced I/O

Chapter 13 - Daemon Processes

Chapter 15 - Interprocess Communication

Notes

man 1 intro: Gives introduction to section 1 of man pages

man x intro: Gives intro to section x of man pages (See intro for all sections (1 to 8) of man pages)

Former Student Feedback

A former student who had been taught these courses by me, wrote me on 22nd March 2014:

These courses (Advanced Unix Programming and Unix Network Programming) went a long way in helping me land my job at Alcatel-Lucent. I had a one-on-one interview with my hiring manager that was entirely on Unix. After joining the company I learned that this person(manager) was a big time 'Unix fan'. It was very satisfying to have done well in that interview. On the job, we completely relied on Solaris Unix based servers and the concepts of processes and threads gained from these course(s), went a long way in helping me grasp the software.
Thank you Ravi Sir.

Sailesh Vasa (2014-03-23 12:41:19)
Thanks for sharing the course material sir. it will be very useful to us as a quick and reliable reference.

APUE - Chapter 1 - Unix System Overview (2014-03-18 09:56)

Given below are references to slides from a US university (links obtained via Google search and so presumed to be freely accessible to anybody on the net) which are the key teaching slides for this topic:

http://www.cs.stevens.edu/ jschauma/810/lecture01.pdf

Self-Study Assignments

Sections 1.1 to 1.3 (Pages 1 to 4): Introduction, Unix Architecture, Logging In, Shells

Section 1.7: Error Handling

Section 1.8: User Identification

Section 1.11 System Calls and Library Functions

APUE - Chapter 2 - Unix Standardization and Implementations (2014-03-18 10:04)

Self-Study Assignments

- Quick Browse through sections 2.1 to 2.4 (Introduction, Unix Standardization, ISO C, IEEE POSIX, Single Unix Specification, Unix System Implementations, Relationships of Standards and Implementations)

- Browse through section 2.5 (Limits, ISO C Limits, POSIX Limits, XSI Limits, sysconf, pathconf, and fpathconf functions, Indeterminate Runtime Limits). Get some understanding of Figures (programs) 2.13 - Print all possible sysconf and pathconf values, 2.14 - Examples of configuration limits (not a program), 2.15 - Dynamically allocate space for a pathname and 2.16 - Determine the number of file descriptors.

- Section 2.7 and 2.8 (Feature Test Macros and Primitive System Data Types)

APUE - Chapter 3 - File I/O (2014-03-18 10:07)

Powerpoint slides and example source file(s) in pdf format: FileIO.ppt, FileIO2.ppt and error.c. Example source file(s) (my own, if I recall correctly, and so different from book example code) in original file format – zip file.

You may find the reference given below to slides from a US university (links obtained via Google search and so presumed to be freely accessible to anybody on the net) worth viewing: http://www.cs.stevens.edu/ jschauma/810/ lecture02.pdf (titled File I/O, File Sharing).

Self-Study Assignments

- Most of the chapter will be covered in class.
- Sections 3.12, 3.13, 3.14, 3.15 and 3.16 (dup, sync, fcntl, ioctl functions and /dev/fd) are Self-Study assignments.
- The entire chapter is part of the course.

Report Submission

Run example programs (in figures) 3.1 - Test whether standard input is capable of seeking, 3.2 - Create a file with a hole in it, and 3.4 - Copy standard input to standard output, and verify that the programs work as expected. Write a report (named C3report.txt) giving your observations. For each example typically limit the observation to two or three lines.

Assignment Submissions

3A) Write a telbook program or set of programs that will provide the following functionality:

1. Allow creation of telbook file with user specified name and size. The size is in terms of number of telbook entries. Each entry will be 40 characters long: 20 characters for Name and 20 characters for Telephone number. (Hint: Use lseek to create file with hole)

2. On user providing index number program should show the entry corresponding to that index number. (Hint: Use lseek to jump to appropriate offset)

3. On user providing index number and entry (name + telephone number) the program should write the entry at the specified index in the telbook.

All system calls that are used in the above program must be checked for error return.

3B) To Telbook program, add log facility. Each action (open, read and write) should (separately) be logged to (same) log file.

The log file name can be hard coded. Program should create log file if it does not exist (atomically).

Log entry format is: Date &Time, Action, content if applicable. The entry should be terminated with a newline (\n).

Log entries should be appended to log file (atomically).

Optional: 3C) Write a program that opens a file using some particular set of file status flags. Query the file status flags and print the file status (using fcntl). Repeat test with different set of file status flags.

APUE - Chapter 4 - Files and Directories (2014-03-18 10:13)

Powerpoint slides and example source file(s) in pdf format: AUP-FilesAndDirectories.ppt, create-file2.c.pdf, createfile3.c.pdf, listdir.c.pdf, readlink.c.pdf, rename.cpp.pdf and suid.c.pdf. Example source file(s) in original file format (my own, if I recall correctly, and so different from book example code) – zip file.

You may find the references given below to slides from a US university (links obtained via Google search and so presumed to be freely accessible to anybody on the net) worth viewing: http://www.cs.stevens.edu/ jschauma/810/ lecture03.pdf (titled Files and Directories) and

http://www.cs.stevens.edu/ jschauma/810/lecture04.pdf (titled File Systems, System Data Files, Time & Date).

Self-Study Assignments

- Entire chapter excluding sections 4.23, 4.24. Also study section 1.3 (Chapter 1).

Report Submission

Study and run example programs 4.3 (Print type of file for each command-line argument), 4.9 (Example of umask function), 4.12 (Example of chmod function), 4.16 (Open a file and then unlink it) and 4.22 (Recursively descend a directory hierarchy, counting file types) and verify that the programs work as expected. Program 4.22 is optional. Write a report (C4report.txt) giving your observations. For each example typically limit the observation to two or three lines.

Assignment Submissions

4a) Write a program that creates a file with ugo+rwx permissions.

4b) Write a program that takes in filenames (with or without a preceding path) as arguments and for each filename prints out the stat info.

4c) Write a program that takes in directory names (with or without a preceding path) as arguments and for each directory prints out the stat info for all the files in that directory.

4d) Write a program backupsrc which takes two arguments: source-directory and destination-directory. This program copies all .c, .h and mak* files from source-directory and its subdirectories (recursively) to destination-directory. Program should exit with an error if destination directory already exists.

Assignments Given in Previous Years – If you have the time you can do them as well

The objective of the assignments is to utilize the system calls that you have studied in programs (different from the example programs). As the time provided is limited I do not expect you to write programs for all the system calls studied (though if somebody does manage to do so, he is welcome).

The programs can be many small programs or a few large programs or any combination of small programs and large programs. A readme file (named C4readme.txt) should be provided which describes in short the program assignments you are submitting.

The evaluation will be based on how well you have explored the facilities provided by the system calls we have studied (I repeat you do not have to include all system calls in your programs). To aid you in choosing these program assignments I have listed below some sample program assignments. You are free to do to your own assignments instead of these.

Sample Assignments

- Write a program that creates a temporary file. (A temporary file is one which gets deleted automatically by the system when the program exits).
- Write a program called issymlink that takes a filename as an argument and checks whether the filename is a symbolic link. If so, it should read the symbolic link file (and not the file pointed to by the symbolic link) and print its contents to stdout. Else it just prints a message saying passed filename is not a symbolic link.
- Write a simple touch program called mytouch which takes a filename as argument. The program should update the access and modification time to the current time.
- Write a program called savecopy which takes arguments as follows:

savecopy dir file1 file2 ...

The program should first create directory dir if it does not exist (otherwise it should go to the next step of copying). Then it should copy all the files specified after the dir argument into the directory dir. If a specified file does not exist it should print an error message and go to the next file. If in directory dir a file with the same name already exists it should give the user an interactive option to overwrite or skip. If any of the arguments after dir are directories the program should print a message and skip the directory from the copy.

- Write a program called lsd which takes arguments as follows:

lsd [dir]

If an argument is passed lsd lists the sub-directories in that directory. If no argument is passed lsd lists the sub-directories in the current directory.

- Write a program called mypwd which prints the current working directory.

APUE - Chapter 5 - Standard I/O Library (2014-03-18 10:15)

Self-Study Assignments

All sections should be read.

APUE – Chapter 6 - System Data Files and Information (2014-03-18 10:17)

Self-Study Assignments

All sections should be read.

APUE – Chapter 7 - Process Environment (2014-03-18 11:47)

You may find the reference given below to slides from a US university (links obtained via Google search and so presumed to be freely accessible to anybody on the net) worth viewing: http://www.cs.stevens.edu/ jschauma/810/ lecture05.pdf (titled Process Environment, Process Control) - see slides from beginning to slide 51, you may ignore, for now, slides from slide 52 onwards as they deal with Process Control which will be covered next in this course.

Reading Assignments

Sections 7.1 to 7.7 (Introduction, main Function, Process Termination, Command-Line Arguments, Environment List, Memory Layout of a C Program, Shared Libraries), 7.10 (setjmp and longjmp Functions); Quick Browse through 7.8 & 7.9 (Memory Allocation, Environment Variables)

Report Submission

Run example programs 7.1 (Classic C program), 7.3 (Example of exit handlers) and 7.4 (Echo all command-line arguments to standard output) and verify that the programs work as expected. Write a report (C7report.txt) giving your observations. For each example typically limit the observation to two or three lines.

Assignment Submissions

7A) Write a program called testexit that takes an integer value from the user and then immediately exits with that value. Write a shell script named calltexit which invokes testexit and after testexit returns prints its return value on standard output.
7B) Write a program called testenv that asks the user to specify an environment variable name. The program should then get the value for this environment variable name and print it on standard output. The program should do this in a loop till the user indicates that he is exiting (in some way you decide). After testing this program to check the standard environment variables, in your (login) shell create another environment variable called TESTEVAR and set its value to "my env data". Now run your testenv program and see whether it can print out TESTEVAR's data. It may not do so in most cases. Why?? How can you fix this? Write this answer also in your C7report.txt file.

APUE – Chapter 8 - Process Control **(2014-03-18 11:51)**

Powerpoint slides: AUP-ProcessTopicsOnly.ppt and AUP-ProcessTopicsOnly2.ppt.

You may find the reference given below to slides from a US university (links obtained via Google search and so presumed to be freely accessible to anybody on the net) worth viewing: http://www.cs.stevens.edu/ jschauma/810/ lecture05.pdf (titled Process Environment, Process Control)
- see slides from slide 52 onwards (on process ids, fork(2), exec(3) wait(2), waitpid(2)).

Reading Assignments

Sections 8.1 to 8.6 (Introduction, Process Identifiers, fork Function, vfork Function, exit Functions, wait and waitpid Functions), 8.8 (wait3 and wait4 Functions), 8.9 (Race Conditions), 8.12 (Interpreter Files); Quick Browse through 8.7 (waitid Function), 8.10 (exec Functions) & 8.11 (Changing User IDs and Group IDs).

Report Submission

Run example programs (figures) 8.1 (Example of fork function), 8.3 + 8.4 (Example of vfork function, Macros to examine the termination status returned by wait and waitpid (not a program)), 8.5 (Print a description of the exit status), 8.6 (Demonstrate various exit statuses) (understand 8.7 (The options constants for waitpid (not a program))), 8.8 (Avoid zombie processes by calling fork twice), 1.5, 8.12 +8.13 (Program with a race condition + Modification of Figure 8.12 to avoid race condition), 8.14+8.15 (Differences among the six exec functions (not a program) + Relationship of the six exec functions (not a program) - so only understand - not run) and verify that the programs work as expected. Write a report (C8report.txt) giving your observations. For each example typically limit the observation to two or three lines.

Also add observations relating to the assignments below in the report. The observations for these can be longer (they they do not have to be longer).

Assignment Submissions

- **8A)** Write a program called mpcalc (multi process calc) that will compare two expressions of the form a^b and c^d and will print out which is less than (or equal to) which. These numbers (a, b, c & d) should be accepted from the user (standard input).

This program should be implemented using multiple processes so that the main process can spawn another process to do the calculation and allow the user to specify another such expression comparison. Instead of the simple a^b and c^d we might have a much more cumbersome and time consuming calculation which would mean that the program will not be able to take the next input till the time consuming expression is calculated. In fact you should simulate a time consuming calculation by making the child (expression calculator) process sleeping for 20 seconds before doing the calculation and printing the result.

The main process should exit when the user indicates that he wants to quit. Accept a maximum of 10 expression comparison commands (exit automatically after the 10^{th} expression comparison is specified).

- **8B)** Copy mpcalc (assignment 8A) as mpcalc2. Instead of child process printing the result, modify it so that child process returns with a status which indicates comparison result.

When the parent process is prompting user for comparison expression input, it should also accept an option to check whether any previous comparison expression has completed and if so to print the result. If no previous process has completed (since last time we checked) then an appropriate message should be printed and the program should go once again to prompt user for input.

If any child process abnormally terminated then the parent process should report the abnormal termination with its termination/exit status. To allow you to test this code, increase the sleep in the child process to a larger value (say 60). Also (in the parent) print out the process id of each child. Now you can send signals to a child process using the kill command from another terminal. Send a child process one of the signals which terminates it and then check whether the parent reports the abnormal termination correctly. [Some signals which usually terminate a process are: SIGINT - 2, SIGQUIT - 3. A signal which always terminates a process is SIGKILL - 9]

Prior to exiting, the main process should wait for all children to complete and report their results.

- **8C)** Write a shell program. It should have the following features:

1. You should be able to execute any commands (programs) with arguments (Figure 1.7 of APUE does not allow any arguments for a command).
2. The termination status and exit status (if any) of the executed program should be reported after termination of the program.
3. Optional: You should allow the user to execute programs in the background.
4. Optional: You should allow user to create / modify environment variables and export them. You should test whether a newly created shell variable which is exported is getting specified in the environment variables of the programs executed from the shell.
5. Any additional features from your side are very welcome :-)

APUE – Chapter 9 - Process Relationships **(2014-03-18 11:53)**

You may find the reference given below to slides from a US university (links obtained via Google search and so presumed to be freely accessible to anybody on the net) worth viewing: http://www.cs.stevens.edu/ jschauma/810/ lecture06.pdf (titled Process Groups, Sessions, Signals)
- see slides from slide 1 to slide 34 as they are on process groups and sessions and relate to Chapter 9.

Reading Assignments

Sections 9.4 to 9.10 (Process Groups, Sessions, Controlling Terminal, tcgetpgrp, tcsetpgrp, and tcgetsid Functions, Job Control, Shell Execution of Programs, Orphaned Process Groups)

APUE – Chapter 10 - Signals (2014-03-18 11:57)

Powerpoint slides: AUP-SingalsTopicsOnly.ppt.

You may find the reference given below to slides from a US university (links obtained via Google search and so presumed to be freely accessible to anybody on the net) worth viewing: http://www.cs.stevens.edu/ jschauma/810/ lecture06.pdf - (titled Process Groups, Sessions, Signals) see slides from slide 34 onwards as they are on signals and relate to Chapter 10.

Reading Assignments

Sections 10.1 to 10.3 (Introduction, Signal Concepts, signal Function), 10.5 (Interrupted System Calls), 10.6 (Reentrant Functions), 10.8 to 10.19 (Reliable-Signal Terminology and Semantics, kill and raise Functions, alarm and pause Functions, Signal Sets, sigprocmask Function, sigpending Function, sigaction Function, sigsetjmp and siglongjmp Functions, sigsuspend Function, abort Function, system Function, sleep Function); Quick Browse through 10.4 (Unreliable Signals), 10.7 (SIGCLD Semantics), 10.20 (Job-Control Signals), 10.21 (Additional Features).

Notes

man 7 signal: Gives details about signals on Linux

Report Submission - Optional

Run example programs 10.1 (UNIX System signals - not a program), 10.2 (Simple program to catch SIGUSR1 and SIGUSR2), 10.4 to 10.8 (Reentrant functions that may be called from a signal handler - not a program, Call a nonreentrant function from a signal handler, System V SIGCLD handler that doesn't work, Simple, incomplete implementation of sleep, Another (imperfect) implementation of sleep), 10.11 to 10.13 (Calling read with a timeout, using longjmp, An implementation of sigaddset, sigdelset, and sigismember, Ways to change current signal mask using sigprocmask - not a program), 10.14+10.10 (Print the signal mask for the process + Calling read with a timeout), 10.15 to 10.21 (Example of signal sets and sigprocmask, Option flags (sa _flags) for the handling of each signal - not a program, siginfo _t code values - not a program,An implementation of signal using sigaction, The signal _intr function, Example of signal masks, sigsetjmp, and siglongjmp, Time line for example program handling two signals - not a program) and verify that the programs work as expected. Write a report (C10report.txt) giving your observations. For each example typically limit the observation to two or three lines.

Also add observations relating to the assignments below in the report. The observations for these can be longer (they they do not have to be longer).

Assignment Submissions

- **10A)- Optional:** Write a program that creates a file and then goes into an infinite loop. Catch the following signals in the program:

a) SIGABRT, SIGBUS, SIGFPE, SIGHUP, SIGILL, SIGINT, SIGIO, SIGPWR, SIGQUIT, SIGSEGV, SIGTERM, SIGTRAP: For these signals program should print a message giving details about the signal caught and then delete the file created in the beginning and finally terminate.

b) SIGALRM, SIGCHLD, SIGCONT, SIGPIPE, SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU, SIGUSR1, SIGUSR2, SIGWINCH: For these signals, print a message giving details about signal caught and then continue with program execution (don't terminate).

For testing the program create conditions (may require program modification for some signals) which generate following signals: SIGABRT, SIGINT, SIGQUIT, SIGCHLD, SIGCONT, SIGPIPE, SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU, SIGWINCH. For testing with other signals use kill command from another terminal.

- **10B**) In the shell program (written previously - assignment 8C) handle Ctrl+C (SIGINT) to terminate current command. Also print more meaningful messages for commands that have been terminated due to SIGSEGV and SIGFPE signals.

- **10C)** In mpcalc2 (assignment 8B) catch SIGCHLD and print termination status as well as output of comparison (if normal termination). Now we do not have to ask user to give a command to issue wait.

- **10D)** Redo mpcalc2 (call new solution mpcalc3) as a worker process pool solution instead of forking everytime a calculation has to be done. mpcalc3 should start off say 5 worker processes at the beginning. Then as the user keys in a calculation task it should be handed off to a free worker process. Use signals for IPC and files for sharing data between processes. If outstanding tasks become more than free processes then the tasks should be queued by mpcalc3 and handed over to worker processes once they become free. Gracefully shutdown all processes when user wants to quit.

APUE – Chapter 14 - Advanced I/O (2014-03-18 11:59)

You may find the reference given below to slides from a US university (links obtained via Google search and so presumed to be freely accessible to anybody on the net) worth viewing: http://www.cs.stevens.edu/ jschauma/810/ lecture08.pdf (titled Advanced I/O / HTTP) - see slides 1 to 19 and 21 to 24, you may ignore slides from slide 25 onwards.

Reading Assignments

Sections 14.1 to 14.3 (Introduction, Nonblocking I/O, Record Locking), 14.5 (I/O Multiplexing), 14.9 (Memory-Mapped I/O).

Report Submission

Run example programs 14.1 (Large nonblocking write), 14.4 (File byte-range lock diagram - not a program), 14.5 (Function to lock or unlock a region of a file), 14.7 (Example of deadlock detection), 14.31 (Example of a memory-mapped file) and verify that the programs work as expected. Write a report (C14report.txt) giving your observations. For each example typically limit the observation to two or three lines.

Also add observations relating to the assignments below in the report. The observations for these can be longer (they they do not have to be longer).

Assignment Submission

- Write one program that uses the select function call. Some possibilities are:

1. Program that does a read with a timeout.

2. Update the program used to test SIGPIPE. Let the writer check whether the pipe is full (using select) prior to writing. If it is full then the writer program should write a message to standard error, sleep for some time and then retry write to the pipe.

3. Modifying program 2 above so that the writer also accepts commands from standard input. Now select will be required to wait for a ready event on either standard input or the pipe.

APUE – Chapter 13 - Daemon Processes (2014-03-18 12:02)

You may find the reference given below to slides from a US university (links obtained via Google search and so presumed to be freely accessible to anybody on the net) worth viewing: http://www.cs.stevens.edu/ jschauma/810/ lecture09.pdf - (titled HTTP, Dæmon processes, System Logging, Shared Libraries) - see slides from slide 19 to slide 28 as they are on daemon processes and logging.

Reading Assignments

Sections 13.1 to 13.3 (Introduction, Daemon Characteristics, Coding Rules), 13.4 (Error Logging), 13.5 (Single-Instance Daemons)

Report Submission

Run example program 13.1 (Initialize a daemon process) and verify that the program works as expected. Write a report (C13report.txt) giving your observations. Limit the observation to two or three lines.

Also add observations relating to the assignments below in the report. The observations for these can be longer (they they do not have to be longer).

Assignment Submission

- **13A)** Write a simple version of the atd (at daemon program) and the at command. Call your atd implementation myatd and the command as myat. myatd should make an entry in the system log for each command executed.

APUE – Chapter 15 - Interprocess Communication (2014-03-18 12:03)

You may find the reference given below to slides from a US university (links obtained via Google search and so presumed to be freely accessible to anybody on the net) worth viewing: http://www.cs.stevens.edu/ jschauma/810/ lecture07.pdf (titled Interprocess Communication).

Reading Assignments

Entire Chapter

Report Submission

Run example programs 15.1 to 15.4 (Summary of UNIX System IPC, Two ways to view a half-duplex pipe, Half-duplex pipe after a fork, Pipe from parent to child - not programs), 15.5 (Send data from parent to child over a pipe), 15.6 (Copy file to pager program), 15.7 (Routines to let a parent and child synchronize), 15.11 (Copy file to pager program using popen), 15.12 (The popen and pclose functions) and verify that the programs work as expected. Write a report (C15report.txt) giving your observations. For each example typically limit the observation to two or three lines.

Also add observations relating to the assignments below in the report. The observations for these can be longer (they they do not have to be longer).

Assignment Submissions

- **15A)** Write a program which uses the pipe system call. Possible programs are: a) Extending the shell program written previously to handle piped commands b) Using pipe as a means of communicating tasks and results between mpcalc main program and worker processes.

- **15B)** Use fifo for implementing communication between mpcalc and worker processes.

- **15C)** Use Message Queues for implementing communication between mpcalc and worker processes.

- **15D)** Use Shared Memory (and Semaphores) for implementing communication between mpcalc and worker processes.

Unix Network (socket) Programming including pthread Programming (2014-03-21 04:46)

This course may have been last taught by me in 2008 in a deemed university in Andhra Pradesh, India. I may have taught this course to/for three or four batches/years. The class size was typically around 14 to 18 students.

Last updated on 23rd March 2014
This is a Lab course I taught for I. M.Tech. (Comp. Sc.). It teaches network client and server programming using sockets and threads.

On completion of the course, most students have a platform from which they can take off to areas like writing an Internet browser client (like Firefox) or even an HTTP server (like Apache). Of course, we get time only to cover the fundamentals and so the student will have to get deeper into these topics on his/her own. But the platform would have been laid. As a by-product the student would have learned pthread programming which enables him/her to do any multi threaded programming tasks (using pthread or other thread libraries). I have been given to understand that multi-threaded programming is useful for multi-core programming and so I have ensured that adequate coverage is done for this topic.

Prerequisites for the Course

C Programming including debugging skills, User level knowledge of Unix (Unix commands like ls, chmod, mkdir, kill etc.) knowledge of file i/o and signal Unix system calls.

Course Book
Unix Network Programming Volume 1, 3rd edition: The Sockets Networking API by W. Richard Stevens, Bill Fenner and Andrew M. Rudoff (http://www.informit.com/store/unix-network-programming-volume-1-the-sockets- networking-9780131411555). Here's the book support site: http://www.unpbook.com/.

Ideal Coverage (subject to time limitations)

1. Quick revision of TCP/IP fundamentals. Almost all of the topics of TCP/IP fundamentals are covered in a Computer Networks theory course done at P.G. level. So we just did a quick revision. Topics revised: Internet, protocols, client-server model, TCP, connection oriented service, UDP, connection less service, packet switching, sequencing, error control, flow control, full duplex, half duplex, protocol stack, protocol headers. IP address, IPv4, IPv6, Port numbers, well known ports, hostnames, dns, Byte ordering, Network byte order, Ethernet, MAC address.

2. Introduction to sockets:

   (a) Socket APIs: sockaddr structure, socket(), htonx, ntohx functions, bind(), inet _aton(), inet _ntoa(), inet _pton(), connect(), listen(), accept(), read(), write(), close()

   (b) Simple daytime socket client and server program.

   (c) Assignment: Writing a Sayings client and server

3. Robust sockets:

   (a) TCP echo client and server. Server uses worker processes for each connection

   (b) Avoiding zombies by handling SIGCHLD signal

   (c) Using select() to wait on multiple file descriptors for I/O. How select enables writing of more responsive client and server programs.

   (d) Shutdown function enabling closure of only Read or Write halves of the connection. How shutdown() enables graceful closure of connection between client and server.

(e) Assignments: Bringing in all above mentioned features, step-by-step, into Sayings client and server.

4. Posix threads (pthreads):

   (a) Basic thread functions: Pthread _create(), pthread _exit(), pthread _join(), pthread _detach()

   (b) Thread synchronization using thread mutexes and condition variables: pthread _mutex _lock(), pthread _mutex _unlock(), pthread _cond _wait(), pthread _cond _signal(); Dangers of multi-threaded program and how to avoid them.

   (c) Assignments: Sayings server implemented as a worker pool using multithreading – two versions, last version uses condition variables for higher efficiency.

5. HTTP:

   (a) Quick introduction to HTTP protocol

   (b) Optional HTTP Assignment: Program which makes an HTTP browser request and displays/stores response from server.

Index of Links to Chapter Wise Course Pages

Please note that, if I recall correctly, I used slides from usually US university links (obtained usually via Google search and so presumed to be freely accessible to anybody on the net) for all of the topics – why reinvent the wheel? However the topics links below clearly specify the sections covered/to be read as reading assignments and the assignments, most, if not all, of which were created by me specially for this course. So I think they give a good framework for students to learn Network (socket) Programming and pthread programming in a semester in college/university environments. [In some years, due to lack of time the pthread programming followed by the http parts of the course got moved to a separate course done after the Network programming course. Two separate courses allows/gave time for the students to attempt/work on most assignments of the course(s) thereby strengthening the knowledge they gained of the topics covered by the course(s).]

Further, please note that, as a first step, I have focused on putting up the course content used by me to teach this course in the deemed university in Andhra Pradesh, India, suitably modified, on this blog. As part of the minor modifications for putting it up on the publicly accessible blog, some errors may have crept in. I have not checked all the modifications for accuracy (due to other demands on my time). If this course (on this blog) does get utilized by students then the errors in the modified part will come to light and will get fixed by me (or others). I think that is a better way of investing my (and others) time for fixing any errors in this course (on this blog).

Introduction to Networking

Introduction to sockets

Robust and Responsive socket programs

Sockets - Miscellaneous

Threads

http

Former Student Feedback

A former student who had been taught these courses by me, wrote me on 22nd March 2014:

These courses (Advanced Unix Programming and Unix Network Programming) went a long way in helping me land my job at Alcatel-Lucent. I had a one-on-one interview with my hiring manager that was entirely on Unix. After joining the company I learned that this person(manager) was a big time 'Unix fan'. It was very satisfying to have done well in that interview. On the job, we completely relied on Solaris Unix based servers and the concepts of processes and threads gained from these course(s), went a long way in helping me grasp the software.
Thank you Ravi Sir.

UNP - Introduction to Networking **(2014-03-21 09:25)**

Theory courses on networking would have covered the important concepts of networking, OSI model and TCP/IP protocol. However, if that has not been done so far or your knowledge of them is fuzzy then here are some references to slides from US universities (links obtained via Google search and so presumed to be freely accessible to anybody on the net) with suggestions on how to study them as well as the particular topics to be studied (within those slides).

1) http://www.cs.cmu.edu/afs/cs/academic/class/15441-f01/www/lectures/lecture01. ppt
Suggested speed: Quick run through

Suggested (slide) topics to study/read/refresh: Internet, Protocol, Network Edge, Network Core, Circuit Switching, Packet Switching

2) http://www.cs.cmu.edu/afs/cs/academic/class/15441-f01/www/lectures/lecture02. ppt
Suggested speed: Quick run through;

Suggested (slide) topics to study/read/refresh: Layering

3) http://www.cs.cmu.edu/afs/cs/academic/class/15441-f01/www/lectures/lecture03. ppt
Suggested speed: Medium

Suggested (slide) topics to study/read/refresh: Applications and Application Layer Protocols, Client-Server Paradigm, UDP, TCP, Port Numbers, Names and Addresses. You may exclude the Socket API detailed slides as that is covered later on in this course.

4) https://cs.nmt.edu/ liu/CSE389/Lect _01.ppt

Suggested speed: Quick run through

Suggested (slide) topics to study/read/refresh: Headers, Router, Byte Ordering, Network Byte Order, MultiPlexing, Modes of Service, Error Control, Flow Control, End-to-End v/s Hop-to-Hop, Buffering, Addresses, Broadcasts

5) https://cs.nmt.edu/ liu/CSE389/Lect _02.ppt

Suggested speed: Quick run through

Suggested (slide) topics to study/read/refresh: Ethernet, CSMA/CD, IP datagrams, IP Addresses, Class A, B.., IP Services, UDP, TCP, Modes of Service, Connection Oriented, TCP vs. UDP. You may exclude the TCP segment detailed slides, the three-way handshake slides, the IP datagram example detailed slides and the HTTP slides.

6) http://www.cs.utep.edu/cheon/cs3331/notes/network.ppt

Suggested speed: Quick run through

Suggested (slide) topics to study/read/refresh: Socket Programming, Server vs. Client Sockets, Server Sockets, Client Sockets, Echo Server, Echo Client. You may exclude Multi Echo Server and the whole set of RMI slides (last part of the slides).

UNP - Introduction to Sockets **(2014-03-21 12:28)**

Given below are references to slides from a US university (links obtained via Google search and so presumed to be freely accessible to anybody on the net) which are the key teaching slides for this topic:

1) Basic Socket API 1: http://www.cs.rpi.edu/academics/courses/-spring98/netprog/lectures/ppthtml/soc kets/sockets.ppt

2) Basic Socket API 2: http://www.cs.rpi.edu/academics/courses/-spring98/netprog/lectures/ppthtml/tcp _sockets/tcp _sockets.ppt

The references to chapters and examples below are of the course book (and not the above slides).

Show/explain the working of:

- intro/daytimetcpcli.c
- intro/daytimetcpsrv.c

Reading Assignments

Chapter 1 Introduction

Chapter 2 Sockets Introduction

Assignments

Assignment 1

Study and try out the daytime tcp client and server example.

Assignment 2

Write Sayings client and server programs with the following behavior

- As in the daytime server, whenever the server accepts a connection from a client, it sends a saying to the client and then closes the connection.
- The client just reads the saying from the server and then displays it.

- Make just one source file each for the client code and the server code. Avoid using the library functions such as readn, writen. The assignment folder containing the source files should be named Saisays.
- The client program should take the ipaddress and port of the server program as an argument in the form of ipaddr:port. e.g. "./saicli 192.0.2.5:4000". This will enable us to test the client program with any Saisays server. The server program should also take the same argument. In the case of the client the argument refers to the server ip address and port. In the case of the server also it refers to the server's ip address and port. This will allow us to copy and run the server program to any machine without program modification and recompilation.

Assignment 3

Write a program called stresser which will subject the server to a stress test by flooding it with requests from multiple clients. Simulate server doing time consuming tasks by adding a sleep for some milliseconds, if required.

Assignment 4

Then increase the 'availability' of the server by creating a separate process for each connection accepted by it. This new process handles the communication with the client.

Also, subject the modified server program to a stress test by connecting to it from multiple clients, initially from the same machine and later on from many different client machines.

Write a report giving your observations and conclusions regarding the assignments. For submission of assignments put solutions for each assignment in a separate folder named appropriately (e.g. assign1).

UNP - Robust Sockets (2014-03-21 12:49)

Typical Server and Client

Show/explain the working of TCP Echo server and client (version 1) from the book source code:

- tcpcliserv/tcpcli01.c
- tcpcliserv/tcpserv01.c
- lib/str _echo.c
- lib/str _cli.c

Robustness and Responsiveness Issues with above server and client

Zombies

Server listener should clean up worker server processes by SIGCHLD handler and waitpid.

Show/explain the following code:

tcpcliserv/tcpcli04.c : Multiple connects to server. On exit results in zombies which are not effectively handled by wait in SIGCHLD handler (needs waitpid in a loop).

tcpcliserv/tcpserv04.c tcpcliserv/sigchldwaitpid.c

(Flawed SIGCHLD handler) tcpcliserv/sigchldwait.c

Server Process Termination

If server is terminated prematurely tcpcli04.c will know of it only on attempting a socket library call. Typically it will be waiting for user input at fgets(). A more responsive client would detect that the server has terminated even while waiting for user input at fgets() and immediately inform the user of the situation. Solution is to use select() function to block on both stdin as well as socket.

Show/explain the following code:

select/strcliselect01.c

Shutdown function

The shutdown function allows us to terminate only read or write directions of the socket. This is useful in scenarios where we want to initiate close from one end but don't want to lose any data that the other end may have sent us but which is in transit. Using close() would simply throw away the data in transit. Shutdown of write end of socket followed by read on read end will ensure proper closure of connection without any loss of data which was in transit at the time of shutdown inititiation.

int shutdown (int sockfd, int howto)

howto: SHUT _RD – Only read half of connection is closed

SHUT _WR – Only write half of connection is closed

SHUT _RDWR – Both read and write halves of connection are closed.

The following version of str _cli function uses shutdown() instead of close(). It also operates on buffers instead of line centric code (e.g. Readline). Show/explain the code.

select/strcliselect02.c

Reading Assignments

Chapter 4 Elementary TCP Sockets

Chapter 5 TCP Client/Server Example

Section 6.3 to Section 6.8 of Chapter 6 I/O Multiplexing: The select and poll Functions.

Assignments

Assignment 1

You should try out all the examples mentioned above and write down your observations.

Assignment 2

Implement what you have learnt about robustness and responsiveness to improve Assignment 4 of Introduction to Sockets. Write down your observations of the same.

UNP - Sockets - Miscellaneous (2014-03-22 05:50)

Line Oriented IO Issues

read functions which buffer data may not work properly with select

readline() function reads a buffer of data and from that buffer returns a line of input back to caller. Additional read data that is available will be kept in readline() function's buffer and returned at subsequent call. Further readline() will block till at least one line of data is read.

Show/explain the following code:

test/readline1.c, lib/readline.c, lib/readn.c

Such a readline function is very useful for network programs dealing with line oriented protocols like HTTP, FTP etc. However there are many issues with such buffered IO functions:

- If select is used to wait for input from multiple sources (say socket and stdin), select may block even though readline() has a line of data (read during earlier invocation of readline where read returned multiple lines in one call). Select uses the stdin fd and is unaware of the internal buffer of readline().
- The second version of readline() above allows for such issues to be fixed by the programmer first checking readline's exposed buffer for data before calling select. But this increases complexity of programming.
- If readn and readline function calls are mixed unexpected behaviour may occur.
- Even if the network programs expect data to be exchanged only in lines, due to bugs or malicious attempts some data may be sent which is not line terminated. Using a function like readline will make it difficult for the network program to detect such data and flag it as an error.
- For the same reasons mentioned above stdio functions like fgets should not be used in socket programs.

Readn function suffers from some of the issues that readline has. So ideally one should avoid using readn type of functions as well.

So what should one do if lines (or fixed amt of data) have to be read (and written)

- Always think in terms of buffers of data being read and written over sockets.
- If a line is expected, read data into a buffer and check the buffer to see whether it contains a line
- If a fixed amt of data is expected typically one has to continue reading till at least expected amt of data has arrived. But this should be done at a top level instead of in a readn kind of function. This way the partially read data is always available in a buffer for the top level code to do any error checking or similar kind of task, instead of the partially read data being hidden away in a readn() routine's private buffer.

Reading Assignments

Section 3.9 (readn, writen, and readline functions)

Section 6.4 to Section 6.7

———————————————————————————-

Socket Options

- Various attributes are used to determine the behavior of sockets.
- Setting options tells the OS/Protocol Stack the behavior we want.
- Support is provided for generic options (apply to all sockets) and protocol specific options.

getsockopt() gets the current value of a socket option.

setsockopt() is used to set the value of a socket option.

int getsockopt(int sockfd,

int level,

int optname,

void *opval,

socklen _t *optlen);

level specifies whether the option is a general option or a protocol specific option (what level of code should interpret the option).

SO _RCVTIMEO Option to be set for so that socket receiving functions timeout after specified time

But man page for socket (7) says that it cannot be set by user on Linux!!!

recv and send functions are also available instead of read and write for sockets.

Reading Assignment

Chapter 7 Socket Options

———————————————————————————————————

Name Address Conversions

DNS provides Name and address conversion.

/etc/hosts file can be used to create some name to address mapping entries so that name address conversion facility in limited form is made available without DNS.

gethostbyname() function returns ip address for name. Reentrent version is also available.

names/hostent.c

gethostbyaddr() function takes ip address and returns hostname

/etc/services file maps service names to ports

getservbyname() gives port number for a service name, protocol name pair

e.g. sptr = getservbyname(“ftp”, “tcp”);

names/daytimetcpcli1.c

gethostbyname, gethostbyaddr support only IPv4. getaddrinfo() supports both IPv4 and IPv6 and handles both name-to-address and service-to-port translation.

Reading Assignments

Chapter 11 Name and Address Conversions

———————————————————————————————————————

Daemons and inetd

Given below is a reference to slides from a US university (links obtained via Google search and so presumed to be freely accessible to anybody on the net) which are the key teaching slides for this topic:

http://www.cs.rpi.edu/academics/courses/spring98/netprog/lectures/ppthtml/ine td/inetd.ppt

Show/explain the following code:

lib/daemon _init.c : Linux has daemon function which does the same.

Inetd run server

lib/daemon _inetd.c inetd/daytimetcpsrv3.c

Reading Assignments

Chapter 13 Daemon Process and the inetd SuperServer

———————————————————————————————————————

Assignments

Assignment 1

Modify the Saisays server and client programs that you have written, so that both of them use name-to-address and service-to-port conversions. The server program should take the service name as an argument and the client should take the IP address/hostname of the server, and the service name as arguments.

Assignment 2

Modify the Saisays server program so that it runs as a daemon process. It should also report appropriate messages to the syslog daemon (the central logging facility of UNIX) using the syslog() call with appropriate priority levels for different types of messages.

The daemon should write to the log all significant events which include messages on the daemon starting up, on the server being brought up, on acceptance of a connection from a client, on the occurrence of an error and anything else you consider important.

Assignment 3

Improve the Saisays server program by making it possible for it to be run as a Superserver based server (i.e. based on xinetd or inetd). It should write appropriate messages to syslog.

UNP - Threads **(2014-03-22 08:23)**

Given below is a reference to slides from a US university (links obtained via Google search and so presumed to be freely accessible to anybody on the net) which are the key teaching slides for this topic:

http://www.cse.unr.edu/ mgunes/cpe401/cpe401sp09/Lecture18.ppt

[The slides that I used when I taught the course could not be located in US university sites easily via Google Search. But I found them at the following two links:

http://www.learningace.com/doc/1205385/e48a70888d6eecc154d027da7ecc4279/threa ds http://www.docstoc.com/docs/111876661/Threads-Programming]

An interesting additional reference is: http://www.cs.fsu.edu/ baker/realtim - e/restricted/notes/pthreads.html

Show/explain the following code:

nonblock/strclifork.c threads/strclithread.c

threads/tcpserv01.c threads/tcpserv02.c

Mutexes

Multi-threaded program which increments a global variable incorrectly ('No Synchronization')

threads/example01.c

Corrected version of above program using a mutex to protect the shared variable

threads/example02.c

Condition Variables

'Web client and simultaneous connections' using condition variables to specify which thread to wait for

threads/web03.c

Client/Server Design Alternatives

TCP Prethreaded Server, per Thread accept()

server/pthread07.h server/serv07.c server/pthread07.c

TCP Prethreaded Server, Main Thread accept()

server/pthread08.h server/serv08.c server/pthread08.c

Assignments

Assignment 1

Modify the Saisays server and client (stresser) programs of Assignment 4 – IntroSockets by making them 'multi-threaded' instead of spawning multiple processes for client connections.

Repeat the stress test against this server and perform a comparison of these results with the results obtained in the previous assignment.

Write a report giving your observations and conclusions regarding the assignment.

Assignment 2

Understand and try out the 'mutexes' and 'condition variables' examples in Chapter 26 (Threads) of the course book.

Understand and try out the different examples of client/server design alternatives examined in Chapter 30 of the book.

Assignment 3

Consider a fictitious Simple File Transfer Protocol (SFTP) which is as follows:

- It is a line oriented ASCII protocol i.e. requests and replies both are in ASCII (not binary) and terminated by \n
- It allows clients to access files in just one directory (called as SFTP data directory) managed by the server. It can list the files in the directory and also get a file. To simplify the protocol we will limit the files in the directory to be only text files (and no subdirectories).
- Requests consist of just request keywords and optional parameters. They are as follows:
  - List\n
  - Get filename\n
  - Quit\n
- Responses consist of a mandatory header and optional body.
- Response Header always has a status line followed by a content length line as follows:
  - Status nnn\n
  - Content-Length nnnnnn\n
- Status code values are as follows:
  - 200 - Means OK
  - 401 - Invalid Command
  - 402 - File not found
- All responses will have status. If the request has been handled successfully then the Status response of 200 is returned otherwise an appropriate Status error response is returned.
- Content-Length line gives the length of the contents that follow after the header. Examples are as follows:
  - Content-Length 0\n - For the case when there is no data that follows. Typically for error status responses and if for List there are 0 (no) files in the SFTP data directory.
  - Content-Length 246\n - For the case when 246 bytes of data follow the header. Typically for List if the string containing the filenames including the return characters is 246 bytes and for Get if the text file whose data follows the header has a file size of 246 bytes.
- List command lists the files in the SFTP data directory. Each filename is on a separate line ended by a \n. If there are no files in the directory then no data is returned to client (but response header is returned).
- Get command gets the file data that it requests. E.g. Get abc.txt\n. If the file does not exist a suitable error status is returned to the client.

- Quit command causes the server to return status of OK and Content-Length 0, after which the server closes the connection.

Write an SFTP server program which implements the application level protocol (SFTP) Simple File Transfer Protocol. (Implement only the 'GET' command).

The implementation should be multi-threaded with the server using prethreading to create a pool of available threads when it starts.

Implement the server program first with the main thread calling accept() and then with each thread in the pool calling accept().

Compare the performance of the two approaches using statistics like the response time of the server.

Write a report giving your observations and conclusions regarding the assignments.

Reading Assignments

Chapter 26 Threads

Chapter 30 Sections 30.11 and 30.12

Additional Reading

Implementing a read-write mutex:

http://doc.qt.nokia.com/qq/qq11-mutex.html

http://stackoverflow.com/questions/1350994/is-it-safe-to-read-an-integer-vari able-thats-being-concurrently-modified-without

http://stackoverflow.com/questions/1087771/do-i-need-to-syncronize-thread-acc ess-to-an-int

Need mutex for multiple read:
http://www.codeguru.com/forum/archive/index.php/t-495831.html

Do I need to lock a mutex for reading a variable:
http://www.openqnx.com/PNphpBB2-viewtopic-t11405-.html

Per-thread data example: http://www.cs.fsu.edu/ baker/realtime/restricted/examples/threads/perthread.c

POSIX Threads programming tutorial: Lawrence Livermore National Laboratory
https://computing.llnl.gov/tutorials/pthreads/

Introduction to Parallel Computing: Lawrence Livermore National Laboratory
(Has a section on Parallel Programming Models where it notes that Threads Model and Message Passing

Model are two of the many models for Parallel Programming)
https://computing.llnl.gov/tutorials/parallel _comp/

UNP - http (2014-03-22 09:16)

Given below is a reference to slides from a US university (links obtained via Google search and so presumed to be freely accessible to anybody on the net) which are the key teaching slides for this topic:

www.cs.rpi.edu/courses/fall02/netprog/notes/web/web.ppt - till slide 39, you may ignore the later slides.

[The slides that I used when I taught the course could not be located in US university sites easily via Google Search. But I found them at the following two links:

http://www.learningace.com/doc/1990348/748ff2abb5f5a9ee6c4d5c822a3d34aa/http

http://www.powershow.com/view/beb4-NjVjY/HTTP _Hypertext _Transfer _Protocol _powerpoint _ppt _presentation]

Additional references:

HTTP Made Really Easy: http://www.jmarshall.com/easy/http/

RFC1945 - HTTP/1.0: http://www.faqs.org/rfcs/rfc1945.html

RFC2616 - HTTP/1.1: http://www.faqs.org/rfcs/rfc2616.html

RFC 2396 - Uniform Resource Identifiers (URI): Generic Syntax: http://www.faqs.org/rfcs/rfc2396-.html

RFC822 - ARPA Internet Text Messages Format: http://www.faqs.org/rfcs/rfc822. html

RFC 1521 - MIME (Multipurpose Internet Mail Extensions) Part One: http://www.faqs.org/rfcs/rfc1521.html

RFC 1522 - MIME (Multipurpose Internet Mail Extensions) Part Two: http://www.faqs.org/rfcs/rfc1522.html

Tutorial - HTTP Proxy - http://www.ragestorm.net/tutorial?id=15

RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication: http://www.faqs.org/rfcs/rfc2617.html

Base 64 Resources

Online encoder/decoder: http://www.paulschou.com/tools/xlate/

Source code for encode/decode: http://www.adp-gmbh.ch/cpp/common/base64.html

---

Assignments

Assignment 1

Write a browser “brow” implementing HTTP **GET** request: brow –d GET ipaddress /……..

The server response should be displayed on stdout. There should be a debug option –d to display (on/off) the reply messages.

Linux Kernel Customization - Mini Course (2014-03-29 12:54)

This mini course was taught by me only once in 2008, if I recall correctly, in a deemed university in Andhra Pradesh, India. As it was not a full-fledged course I have chosen to provide details of the mini-course, and suggestions to make it a full-fledged course, as a report.

Towards the end of the Minix Kernel Internals Lab course for Ist M.Tech. (CS), for a period of 6 days this mini-course/sub-course was conducted. The following topics were covered:

1. Building the Linux Kernel. Students were taken through the steps of building the Fedora 7 Linux kernel. Topics explained included: Need for customized kernels (embedded devices, performance optimization, usage of new kernel features which are available only as kernel patches, bug fixes), boot process, grub, linux kernel executable files, statically linked kernel and dynamically loaded modules, related commands like uname, lsmod, /proc pseudo-filesystem, lspci etc., rpms needed for building kernel, configuring the kernel, installing the newly built kernel.

Students built two to three versions of the Fedora 7 kernel: A standard kernel similar to the Fedora distribution kernel and one or two smaller kernels. Some groups of students were able to reduce the kernel dynamic modules from 441 MB to less than 100 MB. They also were able to marginally reduce

the statically linked kernel. As time was short we stopped at students getting a reasonable idea of what kernel customization involves.

1. Very Simple Hacking of Linux Kernel. Kernel code was modified to include some printk statements. Students rebuilt this kernel and observed the output of their printk statements.
2. Writing simple Linux Kernel modules. Students wrote one or two small modules and inserted them into the running kernel using insmod. A small kernel module which handles interrupts was also studied and demonstrated.
3. Kernel patches. Quick introduction to Linux kernel patches, applying a patch and also creating a patch; importance of patches when contributing to Linux kernel community. We did not have time to do hands-on assignments for these topics (kernel patches).

Note: Evaluation of students was done by studying their assignment reports and by a Viva voce.

Linux Kernel Customization slides

Remarks

While this sub-course of 6 days (2 timetable hours per day) did give students a foot-hold in the rather intimidating area of Linux Kernel customization and Linux Kernel Programming, the period was too short to achieve any substantial goals.

I now feel that the minimum goal for such a sub-course would be the following:

• Build a minimal Linux kernel for an embedded device. Perhaps we could test that kernel using a simulator for that embedded device.

• Write a small but functional Linux device driver.

If these two goals are achieved then students will have gained something substantial. It may be of direct help for a possible device driver IInd M.Tech project. Further, students can mention it in their biodata and also talk about it during job interviews.

However, for such a sub-course the minimal period would be 4 weeks (assuming 8 timetable hours per week).

If required the course can also be made a full course by making the tasks mentioned above more complex like writing a full fledged device driver, and adding suitable topics from the resource links given below.

—- end report —

Making the Linux Kernel For Fedora 7 - Instructions and Log

Resource Links

(IBM Developer Works) Hacking the Linux 2.6 kernel, Part 1: Getting ready - http://www.cagdastopcu.com/wp-content/uploads/2010/01/l-kernelhack1-pdf.pdf

(IBM Developer Works) Hacking the Linux 2.6 kernel, Part 2: Making your first hack - http://marcelotoledo.com/wp-content/uploads/2008/04/l-kernelhack2-a4.pdf

Personal Fedora 7 Installation Guide - http://www.mjmwired.net/resources/mjm-fedora-f7.html

http://kernelnewbies.org/, http://kernelnewbies.org/KernelBuild

http://www.kernel.org

The Linux Boot Process - http://www.linuxhomenetworking.com/wiki/index.php/Quick _HOWTO _: _Ch07 _: _The _Linux _Boot _Process

Modifying the Kernel to Improve Performance - http://www.linuxhomenetworking.com/wiki/-index.php/Quick _HOWTO _: _Ch33 _: _Modifying _the _Kernel _to _Improve _Performance

http://www.linuxfromscratch.org/

Linux Kernel Programming US university course link - http://www.cs.utexas.edu/users/ygz/378-03S/

Minix Kernel Internals **(2014-03-30 07:34)**

This course must have been last taught by me, if I recall correctly, in 2008 in a deemed university in Andhra Pradesh, India. I think I taught this course for two or three batches/years. The class was I. M.Tech. (Comp. Sc.) and class size was typically around 14 to 18 students.

Essentially this was a course designed to give students a feel of kernel internals design and code.

One interesting discussion I had with other faculty was on whether we should have a Minix Kernel Internals course or a Linux Kernel Internals course. The important points related to my view of that discussion are available here.

Prerequisites for the Course

C Programming including debugging skills, User level knowledge of Unix (Unix commands like ls, chmod, mkdir, kill etc.) and theoretical knowledge of modern operating systems like Unix./Linux Kernel Internals. Note that students typically would learn such theory of modern operating systems in a separate theory course done either prior to this course or in parallel to this course.

Course Book

"Operating Systems - Design and Implementation", Second edition by Tanenbaum and Woodhull, http://minix1.woodhull.com/osdi2/.

A later edition, 3rd edition, is available, http://www.flipkart.com/operating-systems-design-implementation-3rd/p/itmdytczc xvugj3h. But I do not know whether it is appropriate to use that book for this course material. [BTW Instructor resources (would be useful for students too, I guess) is available here on request, http://www.pearsonhighered.com/educator/-academic/product/0,3110,0131429388,00 .html #resources. This course site of a US university, http://www.cise.ufl.edu/ nemo/cop4600/, seems to be using the 3rd edition book as the main course book. The site has slides which can be downloaded.]

Note: While the course book has some theory of Operating Systems as well, this course focuses only on design and implementation of Minix. A separate theory course on Operating Systems is done by students either prior to this course or in parallel to this course.

Minix website: http://www.minix3.org/

Teaching with Minix Howto (has links to Minix course websites): http://minix1.woodhull.com/teaching/

Chapter Wise Study Notes

Chapter 1 - Introduction

If time permits the whole chapter should be read. Sections 1.2.5 - History of Minix, 1.3 - Operating System Concepts, 1.4 - System Calls are important sections which MUST be read (they are a reading assignment for this course).

Given below are some notes of the important parts of this chapter for this course.

History of MINIX

- Tanenbaum writes MINIX from scratch using same system interface as UNIX but the implementation being completely different from AT &T's UNIX thereby avoiding licensing issues and making MINIX suitable for study in edcuational institutions (like universities). MINIX stands for mini-UNIX.
- Students can dissect a real operating system - MINIX - like biology students dissect frogs.
- MINIX code is meant to be studied and be readable unlike UNIX where the objective of the design was efficiency.
- This book's MINIX version is based on POSIX standard.
- MINIX is written in 'C' programming language.
- MINIX initial implementation was for the IBM PC; subsequently ported to other machines/architectures like Macintosh and SPARC

- Running MINIX similar to running UNIX; commands like ls, cat, grep and make as well as the shell are available.
- Finnish student, Linus Torvalds, wrote a MINIX clone called LINUX to be a "feature-heavy production system" as against an educational tool that MINIX is.

Operating System Concepts

- System calls are the interface between OS and user programs

Processes

- A program in execution is a process
- A process had address space - memory locations which the program can read from and write to
- Address space is divided into code - executable program, data and stack; (stack is also data but of a different type)
- Each process has a set of registers including program counter and stack pointer needed to run the program (using the underlying processor and processor instructions).
- In a time-sharing system OS suspends one process, saves its process state, and starts/re-starts another process; the suspended process is re-started after some time at which time first its process state must be restored; process table entry stores the process state when the process is suspended
- Command interpreter or shell has to execute programs; as part of program execution and control of that execution it needs a system call to create a process which will execute the program; on the program finishing a system call is needed for the process to terminate

Files

- Broad category of system calls that deal with the file system
- system calls are available to create and remove files; to open, read, write and close files
- hierarchical directory structure: system calls available to create and remove directories; put an existing file in a directory, remove an existing file from a directory; a directory itself can be a file entry in another directory thereby providing for a hierarchical directory structure
- 9-bit file protection code giving rwx permissions for owner, group and others

Shell

- Command interpreter is called shell; it is the primary interface between user on terminal giving commands and the operating system

- On user login a shell is started up for the user; typical prompt is $; user types in command at this command prompt
- input and output file redirection via < and >

Memory Layout of Process

- A process has three segments in its address (memory) space - text (or code), data and stack; data segment grows upward and stack segment grows downward (see fig. 1-11)

Monolithic Operating System with User Mode and Kernel Mode

- Most CPUs have (at least) two modes - kernel mode where all instructions are allowed and user mode where some instructions are prohibited
- Special trap instruction in the CPU instruction set known as kernel call or supervisor call switches the CPU from user mode to kernel mode
- Parameters of the kernel call/supervisor call instruction are used to identify which system call should be invoked

OS components

- Four components of Minix OS - process management, I/O device management, memory management and file management; course book chapters are also on these lines.

Lab work: Installing VMWare on Linux and then installing Minix on VMWare (Instead of VMWare for some years/batches Bochs emulator was also used). One great advantage of using Minix on VMWare for this course was that Minix kernel crash (due to modified kernel code of students typically) only crashed the virtual machine. Bringing up a new virtual machine Minix (with stable kernel) on VMWare was quick and straightforward. This contrasts very positively with the time lost on Minix kernel crash on physical machine and resultant time and effort to bring up Minix (with stable kernel) on the physical machine.

This link provides info. on installing Minix on VMWare and other virtual machine/emulator software, and has some other related support info. too: http://minix1.woodhull.com/hints.html #emul-virt.

Chapter 2 - Processes

Section 2.5 - Overview of Processes in Minix and section 2.6 Implementation of Processes in Minix (pages 93 to 147) are very vital sections of the book. These are the first, and perhaps the most important, sections that delve into Minix kernel source code. These sections have to read and understood in conjunction with the actual source code they refer to.

[The Minix source code listing is provided in the book itself as Appendix A - The Minix Source Code from page 523 to page 903 (380 pages) consisting of 27646 lines of (line numbered) code. Appendix B is an Index to Files (pages 907 and 908) which gives the filenames and starting line number in listing for files in include directory, kernel, memory manager and file system. Appendix C is an Index to Symbols (Pages 911 to 923) giving symbol name and listing line number. These symbols include function names and macros (#define).]

[I discussed these sections (2.5 and 2.6) in class and also walked through appropriate code where required. These were intense sessions and required high level of commitment from students if they wanted to really benefit from the course.]

These sections involve some assembler code. Some, if not all, the detailed pages/sub-sections are listed below:
a) stackframe _s structure: Page 115 Lines 4537 to 4558 (Lines refer to the source code listing included as Appendix A)
b) mpx386.s Page 124
c) Section 2.6.7: Interrupt handling in MINIX, Pages 128 to 137
d) Section 2.6.10: Hardware-Dependent Kernel Support Pages 142 to 145
e) Section 2.6.11: Utilities and the Kernel Library, Pages 145 to 147

The assembler code (if I recall correctly) is Intel 386 code. So there must be some understanding of Intel 386 architecture and instructions. The students would have taken a computer architecture course in the past which would have covered Intel 386 architecture and instructions to some extent, at least.

The wikipedia page, http://en.wikipedia.org/wiki/Intel _80386, seems to give a decent overview of Intel 386.

An Intel 386 programmer's reference manual is available here: http://intel80386.com/. From this reference these two chapters are important for understanding the assembler code (if I recall correctly) in Minix source code referenced by the course book:

- Chapter 2 - Basic Programming Model, http://intel80386.com/386htm/c02. htm
- Chapter 3 - Applications Instruction Set, http://intel80386.com/386htm/c03.htm

Lab. Assignment: Change process scheduling algorithm - I am afraid I don't recall the details now - could be changing scheduling from round-robin to lottery scheduling (random).

Assessment: Ideally assessment for this very intense chapter would involve quizzes/viva-voce to examine a student's understanding of the design and implementation details covered by this chapter. However, I have to be honest and state that very few students of the batch had the time, inclination and/or resolve to get that deep into Minix design & code. Most students chose to focus on doing the lab. assignments where one could do some fiddling with part of the code without having got a full understanding of the design and code covered in this chapter. I could not blame the students much as the time available for this lab. course in terms of both study time and computer lab. time was limited (competition from other courses and other activities that students had to get involved with). Having said this, I must also say that the objective of exposing most students to the complexity of kernel

internals of Minix got achieved with intense coverage in class of some of the design & code covered by this chapter. So, if in future, say for a IInd M.Tech. (CS) project, a bright student chose to do an OS component design & implementation project (like a filesystem implementation/port to Minix), the background exposure would have been provided by this course for the student to take up the challenge.

Chapter 3 - Input/Output

In this course we do not focus much on this chapter. However the following sections must be read as they may be useful (are useful, if I recall correctly) for better understanding of later chapters:

- 3.4.1 Interrupt Handlers in Minix
- 3.8.3 Overview of the Clock Driver in Minix
- 3.10 The System Task in Minix

Chapter 4 - Memory Management

This whole chapter is an important part of this course and so the whole chapter should be read. In particular the following sections must be studied in-depth:

- 4.6.3 Segmentation with Paging: The Intel Pentium
- 4.7 Overview of Memory Management in Minix (including subsections)
- 4,8 Implementation of Memory Management in Minix (including subsections)

The following topics were discussed in depth in class - fork system call implementation; Process scheduling, process table, process memory map and Translation. Viva-Voce/Quiz was conducted to test understanding of these topics by the students.

Lab. Assignment 1: Modify memory allocation algorithm from first fit to best fit.

Lab. Assignment 2: Add a new system call in MINIX (in MM). The system call should take in, say, two parameters, and have a return value. The system call's service may be a very trivial one as the objective is to understand how add a new system call to Minix (and then test it, of course).

Lab. Assignment 3: Add a new system call in MM that gets the memory map for the requested process.

Chapter 5 - Filesystems

Due to lack of time this chapter was not covered. However, if students have the time and inclination this chapter should be read and understood. [A couple of students who underwent this course in Ist.

M.Tech. (CS) took up implementation/port in Minix of a filesystem supported on Linux but not in Minix (2.0). That project involved understanding this chapter and related code very well.]

1.3 April

Minix OR Linux Kernel Lab – Some Thoughts (2014-04-08 11:22)

The following are rather frank thoughts on whether the lab. course should be on Minix or Linux kernel. This comes from a document I had prepared for discussions with other faculty in 2008.

A Ist M.Tech Lab course can be justified from two perspectives:

- How it gives students the skills and knowledge that helps them in good implementation of research projects and IInd M.Tech projects.
- How it gives students the skills and knowledge that helps students to get industry jobs.

Minix Lab course

- Great and perhaps ideal for in-depth O/S teaching at University level
- Much more elegant O/S as compared to Linux
- Wonderful platform for O/S research projects (Linux would be perhaps too cumbersome a platform for O/S research. E.g. New techniques for scheduling may be more easily and perhaps more elegantly tried out on Minix.). However O/S research projects are not so common in Indian UGC/AICTE environment (I believe).
- Very in-depth course. Good students would be able to handle the intensity but average students may not be benefiting much.
- IInd M.Tech. projects may be on Linux or Sun Solaris and not Minix. E.g. Cell processor projects.
- Perhaps only few in-depth OS companies may be more appreciative of in-depth Minix kernel exposure as against overview of Linux kernel and Linux kernel device driver exposure.
- Most companies where students apply for jobs may have preference for Linux Kernel Device driver exposure instead of Minix Kernel exposure
- Minix course should ideally follow course book. This helps students as they can refer to the course book to strengthen their understanding of class sessions.
- Intel architecture exposure would be best suited for the course as the course book follows Intel arch.

- Lab courses are the only courses (barring some minor exceptions) where students get hands-on exposure. A Lab course should focus more on what the student learns to do as against what the student learns to talk about. Minix Lab course seems to be tilted heavily on understanding Minix code as against doing hands-on assignments. Perhaps significantly changing Minix code would take too much time. Existing code changing assignments in Minix seem to be somewhat trivial.
- Ideally Minix code understanding (and even processor and processor related code understanding) could be done as part of related theory courses (O/S or Architecture). But the theory courses may not be in a position to spare the time.
- It would be ideal if we have a defined course content for Minix.

Linux Kernel Lab course

- Will not be so in-depth as Minix
- Would involve Kernel customization, writing kernel modules and device drivers. Some exposure to Linux Kernel internals but lesser than that of Minix.
- May be helpful for IInd M.Tech. projects on Linux platform. Students may be able to explore performance tweaking of kernel. They may even be able to use development versions of the kernel (as they may have new features that their project needs).
- Knowledge of Writing Linux Device drivers may help IInd M.Tech projects involved with devices like OCR scanners, Honeywell sensors etc.
- Knowledge of Linux Kernel customization may help in better utilization of upcoming Cluster computing lab as the OS will most probably be Linux.
- At job interviews most companies in Indian environment will perhaps be more impressed with Linux Kernel exposure as that may be more relevant to work they do (e.g. writing drivers). Most Indian companies employing M.Tech. post graduates may not be into intense OS development.

Making the Linux Kernel For Fedora 7 - Instructions and Log **(2014-04-09 12:59)**

[This post has been created from documents last updated in Feb. 2008]

Making the Linux Kernel For Fedora 7 - Instructions

1) rpm -ivh kernel-2.6.21-1.3194.fc7.src.rpm

(Ignore kojibuilder errors)

2) rpm -ivh sparse-0.4.1-1.fc7.i386.rpm

3) rpmbuild -bp –target= $(uname -m) /usr/src/redhat/SPECS/kernel-2.6.spec
Now have a look at the /usr/src/redhat/BUILD directory. It will have two source trees under a kernelxxx directory.

4) cd /usr/src/redhat/BUILD/kernel-2.6.21/linux-2.6.21.i686

6) If you want to save the current .config then copy/rename it as say .saveconfig. Run 'make mr-proper' (which among other things, deletes .config).

5) cd to configs/ and then copy kernel-2.6.21-i686.config to ../.config OR use saved config file by renaming it to .config

7) Run 'make menuconfig'. Change configuration as desired
If X-Windows is available then you can run 'make xconfig' instead.
See impact of selections here in the remarks on make modules _install (13).

8) Run 'make dep' to set up dependencies. [Seems that this step is not required for 2.6 kernel; was required for 2.4 kernel. But run it anyway.]

9) Run 'make clean' to prepare the source tree for the build.

10) Ensure that Makefile in linux-2.6.21.i686/ has 'EXTRAVERSION=something'. This something is appended to the kernel file that is created and so helps in differentiating between current kernel and newly created kernel. This something is also used as a library name for the modules created in the new build.

11) Run 'make bzImage' to make the new kernel (vmlinux) [Takes a long time]
Check the size of the vmlinux file (kernel file) that has been created.

12) Run 'make modules' to make the new modules. [Takes a very long time]

13) Run 'make modules _install' to install new modules in proper path (/lib/modules/<kernelversion>/kernel/drivers
For a config where no changes were made while running make menuconfig, the /lib/modules/<kernelversion>/ directory reported a size of around 441 MB whereas the installed FC7 kernel on my VM (probably default options) had a /lib/modules/<kernelversion>/ directory of size around 52 MB.
And, obviously, the 'make modules' command for 'no changes while running make menuconfig' config took a huge amount of time. Maybe 1.5 to 2 hours.
So it is very advisable to spend some time at the 'make menuconfig' stage and drop the obviously unnecessary modules (for our needs) from the kernel (modules) build.

14) Run 'make install' to copy new kernel and its files to proper location.
For my kernel build, the command reported 'end _request: I/O error, devfd0....' but it seemed to have copied the kernel to the /boot location. So things may be OK even with the reported error.

15) Verify that /boot directory contains a vmlinuz (compressed version of vmlinux) file whose filename contains the custom kernel version number (picked up from EXTRAVERSION of Makefile)

16) Verify that /boot directory contains an initrd file whose filename reflects the custom kernel version.

17) Verify that /boot/grub/grub.conf has an entry for the custom kernel

Now you are ready to reboot. From the boot (grub) menu choose the custom kernel and try it out.
Note that you can use 'uname -r' command to confirm that your custom kernel is running

Log of other attempt(s)

Later I created another kernel but a minimal one. Went through same procedure on existing .config file (and so of prep kernel). Started with 'make mrproper' step. Unusually though 'make menuconfig' on existing .config file showed a sensible set of options chosen (as against almost all options for prep kernel). I expected existing .config file to show almost all options selected as that is how I ended up with 441 MB of modules!! Gave kernel suffix as min1 (in Makefile as well as in one of the 'make menuconfig' options. The .config file has been saved in the FC7 directory as .config.min1 (To be done)

a) 'make bzImage' took 10 minutes.
b) 'make modules' got over very fast (Maybe less than 3 minutes)!!!! Only a few modules (smb, mymodule etc.) got made. When I checked the .config file the =m option was only for SMB _FS. Maybe that's why it got over so fast. Have still not understood how the config file is so vastly different from the first (prep) kernel one, though I don't seem to have made any significant changes while running 'make menuconfig'. SMB _FS was one of the changes and it got reflected in the build.
When I ran 'make mrproper' it gave some CLEAN xxx messages and took quite some time. 'df' reported much higher free space after 'make mrproper' finished. So I do think that old modules would have got deleted as part of 'make mrproper'. The 'make clean' step ran very quickly (probably because 'make mrproper' had done the cleaning).
c) 'make modules _install' finished in a few seconds. It created a dir /lib/modules/2.6.21-min1min1 (I guess min1 appears twice due to Makefile entry as well as config entry).
/lib/modules/2.6.21-min1min1 occupies just 136 KB!!!!.
Contrast this with /lib/modules/2.6.21-1.3194-fc7 occupying 57268 KB.
And with /lib/modules/2.6.21-prep occupying 441108 KB.
d) Booting into new (min1min1) kernel failed!!!! After uncompressing Linux it gave an error:
'powernow-k8: BIOS error - no PSB or ACPI _PSS objects'
and then just froze.
The powernow-k8 error appears on other kernel boots (checked with prep kernel) as well. So that is not be the problem.
Maybe I need to do a kernel rebuild from the rpmbuild step and not the 'make mrproper' step.

Noticed that 'make mrproper' deletes .config file!!!! So the copy from configs/kernelxxxi686.config is useless. Running 'make menuconfig' creates a .config file with those minimal options. Last time I had changed some entries. This time I will leave it untouched and try building kernel again.
Failed again. But this time there was a message prior to the powernow-k8 message saying something about processor not being Intel. I need to check the config file.

Config file mentioned processor as Pentium III. Changed it to AMD in make menuconfig and am trying 'make bzImage' again.
Failed again with same message: (microcode: CPU0 not a capable Intel processor followed by

powernow-k8 message and then freeze).

I guess I need to repeat the steps right from step 1 (rpm install of source followed by rpmbuild).

Deleted min1 kernel files and /lib/modules files. Emptied redhat/BUILD and redhat/SOURCES and redhat/SPECS directories.
Am now starting at step 1.
Step 3 (rpmbuild) creates a .config file in BUILD/kernxxx/linuxxxx directory. diff between this .config and configs/kernxx-i686.config gave only one extra line in former (# i386). Renamed .config to .config.orig and carried out step 5 (copying configs/kernxx-i686.config as .config)

'make mrproper' deleted .config file!!! So earlier step 5 work was undone!!! So maybe this is the problem step. RH9 kernel needs this but the IBM tutorial for FC7 does not mention this step.

'make menuconfig' gave a set of options (maybe its default as there was no .config file). Saved these options. diff between new .config and .config.orig threw up lots of difference. One key difference was that .config had CONFIG _MPENTIUMIII=y whereas .config.orig had CONFIG _M686=y.
I think this is the incorrect specification that tripped up the earlier min1 and min1min1 builds.

Am deleting .config and copying .config.orig as .config. then will run 'make menuconfig' again.
Yes. The options in menuconfig match with what I had seen in the prep build. So looks like I may not have done the 'make mrproper' step while doing the prep build.
Need to reduce selections (else will have 441 MB of /lib/modules). Plan to try mv .config as .config.orig and then run 'make menuconfig'. In default .config choose 686. Maybe this will fix problem. (Earlier I had tried AMD instead of PentiumIII - that did not work - see above notes). 686 choice is not there in menuconfig - instead the choice is for 586 or 686 which I chose and that resulted in CONFIG _M586=y spec.
Changed Makefile EXTRAVERSION to min1, make dep, make clean
Now am running make bzImage
Then ran Make modules, make modules _install (0 modules created).

make install tripped up towards end as it attempted to write to fd0. I think Floppy needs to off in VMware while running make install.

System got hung and on restart as there was a min1 option, I tried it out. No luck got same error as earlier.
Am going to try to do make install again with Floppy off.
No luck again!!! Same problem.

Next option is to use the configs/kernxxxi686.config and then run menuconfig on it. This configuration is a safe working starting point. I can then deselect items that I am sure are not needed like IrDA and Bluetooth. I think this approach has a better chance of working. Perhaps using menuconfig without any config file gives it a default set which may not be appropriate for my machine.

Tried this option out at college. /lib/modules/xxxprep takes only about 95 MB.
System goes beyond eariler point but does not find /dev/root and results in 'kernel panic.'

Back to Home.
'make defconfig' creates a .config file with default values. Tried it out. The .config file created had much

lesser options that configs/kernxx.i686.config. Processor chosen is Pentium III. Am trying make with the default config now.
Oops!! Got the same error as when 'make menuconfig' created the .config file. Makes sense as defconfig option would be what make menuconfig runs to create a base .config if a .config is not present.

BTW found that menuconfig option PentiumPro corresponds to M _686

Noted that /boot has a config file for fc7 install build. That is the same (but for 2 lines of comments) as configs/kerxxxi686.config!!!! Further the prep kernel (whose config was perhaps an exact copy of configs/kernxxxi686.config) is 1883668 bytes and the fc7 kernel is 1883604 bytes. A difference of just 64 bytes. The /lib/modules significant size difference could be due to installation options where only required modules got copied from install media to hard disk.

That means that the better way to customize kernel would be to start with configs/kernxxi686.config and try to remove some entries (as against using defconfig). Tried it out. Reduced many modules but allowed all LVM/RAID modules. Removed only modules that seemed very safe to remove. Reboot into new kernel worked!!!! Finally. Saved the config file as .configmin1-1W. vmlinuz is 1.59 M as against 1.88 M of both prep and fc7. /lib/modules is 276M as against 441M of prep and 57M of fc7 (But I think fc7 lib size is small due to some modules not being copied from distro media to disk).

Now will try to reduced some more modules.

Tried make bzImage after changing config via make menuconfig. make bzImage seems to be compiling everything. I guess changing the config file implies that all files should be made again. To be on safer side I think I should do 'make clean' after every config file change. To save time I have made quite a few reductions to the config file. Hope the new kernel runs.
It does!! vmlinuz is 1.50M and /lib/modules/xxxmin1 is 215M. Saved config file as .configmin1-2W
Startup threw up errors for bluetooth and something else (as its drivers are not included). some rpc stuff also gave an error but cannot link it to a missing driver. Ran ntsysv to not start bluetooth, nfslock, sendmail etc. at startup.

Have copied config files to guest and then Windows partition.

Now will try for a final reduction (am running out of time and further now I have got enough hang about it to teach the students). Worked!!!
vmlinuz is 1.42 M and /lib/modules/xxxmin1 is 72M (only)!!. Saved config file as .configmin1-3W.
Startup threw some more errors this time around. IPv6 module has been dropped and so some startup programs failed. IPv4 is present and X-Windows operates fine. Further I have been able to ftp between XP host and OS running this kernel.

There definitely is scope for further reduction in modules. But I guess I need to stop now. An interesting possibility for trying out reduction is to go through /lib/modules/xxxmin1 and check out possible candidates for reduction. To confirm possibility of reduction we can check against lsmod which will give us currently loaded modules. If it is not present there (and also does not seem to an obviously necessary module like ext3fs) then the file (xxx.ko) can be renamed and then the system rebooted. If the system starts up properly then maybe that module can be dropped from config (this assumes that kernel worries only about modules it needs during start up). This way avoids a time consuming kernel rebuild.
A comparison with lib/modules of fc7 may also help in identifying required modules. modinfo command

helps in getting detailed info about modules (man modinfo for more details). In fact now I feel this technique can be very promising to reduce modules to way below 57 MB of fc7.

Note: For failed boots I should have looked at dmesg (System log). System log has more info that what is put out on the screen during boot.

Miscellaneous Info.
0) uname -r gives kernel release. man uname for more info
a) free command shows free physical memory.
b) top command gives free memory and some more details.
c) arch command outputs processor architecture
d) Read /usr/src/redhat/BUILD/kernel-xxx/linux-xxx/Documentation/HOWTO. It is a quick intro for a wannabe Linux Kernel Hacker (Developer :-)).
Another must read is kernel-docs.txt in the same directory.
e) lsmod, insmod, modprobe, modinfo are module related commands. man them for info.
lsmod followed by modinfo module-name is a good combo.
f) /etc/modprobe.conf has some module related stuff
g) man bootparam gives info about boot parameters.
h) ???To UPDATE for FC7???kernel customization can reduce vmlinuz size to less than 1 MB. Running system takes up about 5 MB less with casual reduction during customization and /lib/modules/kernel-xxx goes down from 29 MB to around 12 to 13 MB.
Non GUI RH9 can run in 32 MB RAM quite comfortably. ?????
i) In /etc/inittab set default runlevel to 3 for console linux and 5 for GUI linux.
j) cat /proc/cpuinfo is an interesting command. Try it out.
k) lspci -v (another interesting command: lists pci devices)
l) dmesg | more shows the boot up messages.
m) cat /proc/interrupts shows enabled interrupts. man proc details the information accessible from the /proc pseudo filesystem.
n) ntsysv is the text mode command for controlling services startup/configuration.
o) Read /usr/src/linux-xxx/Documentation/CodingStyle

Making FC7 modules - log

Note: The directory FC7/kern-drivers-misc (in the directory containing this file) contains the modules source code and Makefile referred to in the Notes below. Their state is as it was after all notes (points) had been carried out.

1) On FC7 kernel will all modules in default i686 config, tried out mymodule.c module suggested in IBM Linux Kernel hacking tutorial. Notes are:

a) Have to comment out #include <linux/config.h> for successful compilation
b) 'make modules' takes some time (around 10 to 15 minutes maybe). Perhaps the large number of modules selected has something to do with the time taken. Even if the other modules are not compiled maybe something else needs to be done which takes time.
c) make modules success results in creation of mymodule.ko module file.
d) The SUBDIRS=xxx argument to make did not work. So either one can say:

i) 'make modules' at the top of the build tree, OR
ii) make -C /usr/src/redhat/.... modules

e) Was able to successfully load mymodule and unload it as well.
2) Now am trying referring to variable in other kernel code from mymodule.c. Notes are:

a) After adding code to kernel (printk.c - addition of my _variable global variable), compiled kernel with 'make bzImage'
b) To install new kernel used 'make install' and noted that new kernel image in /boot has current timestamp.

old version of this kernel is stored with .old suffix in /boot
grub.conf does not have additional entry for xxx.old linux kernel, as expected.

c) rebooted into new kernel. Confirmed that code change is reflected by using:

i) grep my _variable System.mapxxxx (in /boot directory)
grep lists the symbols in the new System.mapxxxx file but does not list it in the Sytem.mpaxxxx.old file

d) Added code to mymodule.c (which refers and increments my _variable symbol)
e) Ran 'make modules' (took 10 minutes).
f) Loaded & unloaded the module a few times. The my _variable kernel variable was getting changed as expected.
3) Wrote myirqtest.c as instructed. Notes are:

a) Compilation failed. Was able to correct compilation errors by looking at other files in drivers/misc directory. The fixes are:

i) module _param(varname, int/charp, 0644)
instead of MODULE _PARM(varname, "i"/"s");
ii) myinterrupt function does not take the third struct pt _regs argument. i.e. only first two args (int, void *)

b) Test with irq 18 (interface eth0) succeeded. We can use any interface name, say Saieth0. cat /proc/interrupts then lists it as well.
c) Test with floppy irq (7) and timer irq (0) resulted in insmod failure with mismatch error message.
d) Test with i8042, irq (1) succeeded. Seems to be related to keyboard as pressing a key seems to fire two interrupts.
e) Ran startx to see if more interrupts are listed under /proc/interrupts once X is in operation. X did not seem to add new interrupt (handlers).
f) Under X, tested with i8042, irq(12) which seems to be mapped to the mouse. Test succeeded. Moving mouse resulted in print messages. (Had to use dmesg | tail to see the messages under X, whereas on console the messages appeared on console itself and could additionally be viewed using dmesg | tail).
g) Exited X and retested with i8042, irq(12). Succeeded again and confirmed that irq 12 was mapped to mouse.
4) Modified myirqtest.c to print interrupt count (received by this module) as well. Notes are:

a) Took only 7 minutes to 'make modules'.
b) Tests succeeded - now the count is also printed and thereby makes it easy to know the last (10th)

interrupt print message.

———— Old Making RH9 Kernel ———-
1) Check whether /usr/src/linux-2.4 exists.
If not then check whether kernel-source package has been installed using:
rpm -q kernel-source
If this command does not list any kernel-source package then install the kernel-source package using:
rpm -ivh kernel-source-2.4.20-8.i386.rpm

2) cd /usr/src/linux-2.4

3) From /usr/src/linux-2.4/configs copy kernel-2.4.20-i686.config to /usr/src/linux-2.4/.config

4) Run 'make mrproper'.

5) Run 'make menuconfig' (from /usr/src/linux-2.4 directory). Change configuration as desired
If X-Windows is available then you can run 'make xconfig' instead.

6) Run 'make dep' to set up dependencies. [Takes some time]

7) Run 'make clean' to prepare the source tree for the build.

8) Ensure that Makefile in /usr/src/linux-2.4 has 'EXTRAVERSION=something'. This something is appended to the kernel file that is created and so helps in differentiating between current kernel and newly created kernel. This something is also used as a library name for the modules created in the new build.

9) Run 'make bzImage' to make the new kernel (vmlinux) [Takes a long time]
Check the size of the vmlinux file (kernel file) that has been created.

10) Run 'make modules' to make the new modules. [Takes a very long time]

11) Run 'make modules _install' to install new modules in proper path (/lib/modules/<kernelversion>/kernel/drivers

12) Run 'make install' to copy new kernel and its files to proper location.

13) Verify that /boot directory contains a vmlinuz (compressed version of vmlinux) file whose filename contains the custom kernel version number (picked up from EXTRAVERSION of Makefile)

14) Verify that /boot directory contains an initrd file whose filename reflects the custom kernel version.

15) Verify that /boot/grub/grub.conf has an entry for the custom kernel

Now you are ready to reboot. From the boot (grub) menu choose the custom kernel and try it out.
Misc
——–
0) uname -r gives kernel release. man uname for more info

a) free command shows free physical memory.
b) top command gives free memory and some more details.
c) arch command outputs processor architecture
d) /usr/src/linux-xxx/Documentation/modules.txt gives good info about modules
e) lsmod, insmod, modprobe, modinfo are module related commands. man them for info.
lsmod followed by modinfo module-name is a good combo.
f) /etc/modules.conf has some module related stuff
g) man bootparam gives info about boot parameters.
h) kernel customization can reduce vmlinuz size to less than 1 MB. Running system takes up about 5 MB less with casual reduction during customization and /lib/modules/kernel-xxx goes down from 29 MB to around 12 to 13 MB.
Non GUI RH9 can run in 32 MB RAM quite comfortably.
i) In /etc/inittab set default runlevel to 3 for console linux and 5 for GUI linux.
j) cat /proc/cpuinfo is an interesting command. Try it out.
k) lspci -v (another interesting command: lists pci devices)
l) dmesg | more shows the boot up messages.
m) cat /proc/interrupts shows enabled interrupts. man proc details the information accessible from the /proc pseudo filesystem.
n) ntsysv is the text mode command for controlling services startup/configuration.
o) Read /usr/src/linux-xxx/Documentation/CodingStyle

Java Web Programming (including HTML) - 2005 Course Report (2014-04-10 13:26)

This course was taught by me only once in 2005, if I recall correctly, for IInd M.Sc. (Maths) students in a deemed university in Andhra Pradesh, India. Unfortunately I have not been able to locate a record of the assignments given to the class. Therefore I decided to provide a course report using whatever data I have of the course and used my recollections as well as thoughts as I went through the course book and course book slides to add suggestions for assignments as well as a mini-project at the end of the course. [I clearly recall that students did do a mini-project at the end of the course which most, if not all, students found to be very useful in giving them confidence to do future small shopping-cart type web applications using Java servlets and JDBC+database.]

Pre-requisites for the course: It so happened that students of this course had studied C and C++ languages but without much exposure to exception handling in C++. Students also had studied a database theory course and an SQL lab. course. So C++ or other object oriented language, relational database theory and SQL are pre-requisites for this course.

Main Course Book: Core Web Programming, Second Edition by Marty Hall, Larry Brown. This book is now obsolete as can be noted from its website here: http://www.corewebprogramming.com/. Pdf versions of the book may be available on the web. I must also state that the chapters of the book that are used in this course seem to me to be relevant even today. But as I have not taught Java Web programming after 2005 and not had any other reason to keep up with this area, I cannot be absolutely sure about it.

The HTML part (excluding HTML Forms) of the course is done first. Here is the related material.

Next we study those parts of core Java that are different from C++, note the important parts of core Java that are similar to C++, study HTML Forms, and study web server-side Java programming, using the main course book.

- Chapter 6. Getting Started with Java. Slides in pdf format: http://notes.corewebprogramming.-com/student/Java-Introduction.pdf
- Chapter 7. Object-Oriented Programming in Java. Slides in pdf format are http://notes.corewebprogramming.com/student/Java-OOP.pdf and http://notes.corewebprogramming.com/student/Java-Advanced-OOP.pdf. As students already have gone through a C++ course this chapter can be browsed through quickly but with a focus on differences between Java and C++ and explanation of new and different constructs and programming techniques in Java. The latter include:
  - Object being topmost ancestor for all classes
  - Different main function and utility classes and methods like System.out.println()
  - all objects allocated on heap; garbage collection solving the grave memory leak problem; Java using reference instead of pointer
  - Interfaces in Java being a very different and very interesting concept
  - CLASSPATH
  - packages
  - JavaDoc and comments
- Chapter 8. Basic Java Syntax. Slides in pdf format: http://notes.corewebprogramming.-com/student/Java-Basic-Syntax.pdf. This chapter can be browsed through but the following topics in it need proper study. [Please note that at the time I taught this course the students had not had much exposure to vector. list, map and exception in C++. So, if I recall correctly, I taught at least some of them in detail]:
  - Vector
  - Hashtable
  - Exposure to other collection classes of type sets, lists and maps.
  - Primitive data types being wrapped by corresponding wrapper class; parseXxx methods of such wrapper classes to convert string to underlying data type
  - Exceptions: try, catch, finally, throws, throw statements/keywords and exception class hierarchy.
- Chapter 18 HTML Forms. Slides in pdf format: http://notes.corewebprogramming.-com/student/Servers-and-Forms.pdf. The following topics have to be studied:
  - Simple FORM with GET and POST methods and how it used on a browser; Action attribute of FORM specifying URL/CGI program that will receive FORM data; URL encoding
  - CGI (Common Gateway Interface) and examples of it were covered in the course but as it seems to have become obsolete I suggest that it be omitted

  - TEXT controls including password and textarea controls
  - Push buttons including Submit button
  - Check Boxes, Radio Buttons
  - Combo Boxes, List Boxes
  - Server Side Image Maps
  - Hidden Fields
  - File Upload Control
  - Grouping controls

- Chapter 19: Server-side Java: Servlets. Slides in pdf format: http://notes.corewebprogramming.-com/student/Servlets.pdf. This is a vital chapter for Java Web programming. The following topics have to be studied:

  - Servlets get data from FORMs and send output data (e.g. HTML) to client (e.g. browser); Servlets can send dynamically generated webpages to client; servlets run on server and so may have access to server data
  - Advantages of servlets over CGI
  - Free servlet engines (e.g. Apache Tomcat) [I think I used Apache Tomcat for this (student) class.]
  - Hello World servlet; compiling and invoking servlets
  - Servlet generating HTML
  - Servlet life cycle - init(), service(), doGet(), doPost() and destroy() methods
  - Debugging servlets
  - Reading form (query) data in servlets (also called request data); name-value pairs, HttpServletRequest class and its getParameter(), getParameterValues(), getParameter-Names() methods; example servlet program echoing parameters back to client
  - Reading HTTP request headers; getHeaderNames(), getHeader(), getContentLength(), get-ContentType() methods of HttpServletRequest class; HTTP Request headers - Accept, Authorization, Connection, Cookie, Host, If-Modified-Since, Referer, User-Agent
  - Creating the HTTP response; HTTP status code, setStatus(), Common status codes: 200 - OK, 401 - Unauthorized, 404 - Not Found, setHeader(), setContentType(), setContentLength(), sendRedirect(), addCookie(), Servlet example (Prime Number generation)
  - Session Tracking: Stateless HTTP connection, sessions provide means of storing state associated with a client on the server using cookies or URL-rewriting, HTTPServletRequest class methods - getSession(), Hashtable-like mechanism to hold session data, setAttribute(), getAttribute(), isNew(), shopping cart and access count examples for session data

- Chapter 22, JDBC. Slides in pdf format: http://notes.corewebprogramming.-com/student/JDBC.pdf. This is a vital chapter from the point of view of database interaction with servlets (and JSP). The following topics have to be studied:

  - JDBC API - a standardized API to interact with (access) relational databases, JDBC Driver Manager, java.sql package
  - JDBC data types and their mapping to Java data types

- Using JDBC: Loading JDBC driver, defining connection URL, establishing database connection - Connection class, creating a database statement - createStatement() method of Connection class returning object of Statement class, executing SELECT sql statements using executeQuery() method of Statement class which returns object of ResultSet class, executing INSERT, UPDATE and DELETE sql statements using executeUpdate() method of Statement class, Processing ResultSet (in short, details to be covered later), closing the database connection using close() method of Connection class, example code doing all these operations
- ResultSet class: next() method to (attempt to) get next row, getXxx() method where Xxx is a Java type (int, double, string, short, long etc.) which takes in a column Name or column index and returns the associated value for the specified column in the current row in the ResultSet, MetaData methods like getColumnCount(), getColumnName();
- PreparedStatement class for parametized SQL, CallableStatement class for database stored procedures
- SQLExceptions
- Transactions using JDBC

- Chapter 20, Java Server Pages (JSP): Quick introduction to JSP (No time for detailed study). Slides in pdf format: http://notes.corewebprogramming.com/student/JSP.pdf

Class Assignments and Mini-Project

As students go through the various chapters of the book given above (and the HTML tutorial), students are expected to try out the examples given in these chapters (and HTML tutorial), make some small variations, demonstrate them to the teacher and provide a report of these activities. The assessment of this part of the course would be based on these demonstrations and reports.

The main hands-on work in this course should be a mini-project for which significant time, say around a quarter of the time period of the course, should be provided. The mini-project should involve teams of four or five students doing a mini-project together, but with each student providing a separate report detailing his/her contribution to the mini-project. The mini-project problem is left to the students but the scope is controlled by the teacher to make it doable in the time allotted for the mini-project.

One example of a suitable mini-project is a hostel food stall with database tables containing the details of various food items on offer, their availability, their rates etc. and table(s) to hold orders from students as well as track the status of the order. The web interface of the food stall must be attractive but driven by the underlying database data. Session tracking should be used to handle a shopping cart for students that place an order on the food stall and then track the order's fulfillment status.

HTML (2014-04-12 10:48)

The MCLI HTML on-line tutorial, http://www.math.unm.edu/writingHTML/tut/ lessons.html, is the main course material that we will be using to learn HTML (the course book chapters on HTML can be viewed as an additional and optional reference). Please go through the following lessons of the MCLI tutorial throughly: Lessons 0 to 8, Lessons 19 to 22. You may browse through the other lessons.

You may also find these Core Web Programming book slides (in pdf file) on HTML-Introduction interesting: http://notes.corewebprogramming.com/stud ent/HTML-Introduction.pdf.

Given below are some important tags of HTML:

1. Basic Tags:
   (a) < !DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN// >
   (b) < html ... < /html >
   (c) < head ... < /head >
   (d) < title ... < /title >
   (e) < body ... < /body >
   (f) < p > ...< /p >
   (g) < !- ... – >
   (h) < hr >
   (i) < br >
   (j) < b >
   (k) < i >
2. Lists:
   (a) < ol >
   - list item
   - list item
   - ...

   < /ol >

   (b) < ul >
   - list item
   - list item
   - ...

   < /ul >

3. Graphics: < img > ... < /img >
   (a) Attributes for img:
      i. src="..."

ii. align="..."
iii. alt="..."
iv. height="..."(in pixels)
v. width="..."(in pixels)

4. < body > Attributes:
    - bgcolor = #Hexstring
5. Anchors: < a href = "..." > < /a > Variations:
    - < a href = "filename.html" > < /a >
    - < a href = "filename.gif" > < /a >
    - < a href = "http://..." > < /a >
    - < a name = "NAME" > Text to link < /a >
    - < a href = " #name" > Text to activate < /a >
    - < a href = "filename.html #NAME" > Text to activate <
    - < a href = "filename.gif" > < img src="filename.gif" > Go to Document x < /a >
    - < a href = "mailto:enail-id" > Send an email to < /a >
6. Special Characters:
    - &lt
    - &gt
    - & amp
7. Special Effects:
    - < blink >
    - < marquee >
8. Elements of Style:
    - < font > Text goes here < /font > Attributes:
        - size = N (1 <= N <= 7)
        - size = +N
        - base font size=N
        - color = #Hexstring
        - sup
        - sub
        - face = "Font Name(s) (list)"
    - < hr > Attributes:
        - size = N
        - width = x %
    - < p > Attributes:
        - align = center | left | right

9. More Graphics...
    - < img src = "filename.gif | jpeg etc." > Attributes:
        - alt = "Text to be displayed"
        - width = N
        - height = N
        - vspace = N
        - hspace = N
10. Tables:
    - < table > ... < /table > Attributes:
        - border=X
        - cellpadding=X
        - cellspacing=X
        - bgvolor= #Hexstring

Elements of Table:

- < tr > ... < /tr > Table Rows.
- < td > ... < /td > Table Data. Attributes:
    - colspan=X
    - Rowspan=X
    - background="filename.gif | jpeg etc.,"
    - width=x %

11. Images : with & without borders.
12. Bullets for UnOrdered Lists: type attribute.
13. Image Maps:
    - < map name="..." >
    - usemap:" #..." (to be used as an attrubute in the img tag)
    - < nomap >
14. Meta Tags:
    Attributes: Refresh, Content Descriptive Tags
    - < name="Decription" content="..." >
    - < name="keywords" content="..." >
15. Anchors: < a href="URL" target="window-name" >
    (a) Attributes
        i. _top
        ii. _blank
        iii. < base target="window-name">
16. Frames: < frameset rows=x % | N-pixels >

(a) < frame src="URL1" >

(b) < frame src="URL2" >

(c) < /frameset >

(d) < NOFRAMES > Text to be displayed < /NOFRAMES >

(e) Nested Frames

(f) Attributes for frames

i. scrolling=no

ii. name="window-name"

17. Javascript: < script language="Javascript" > THE SCRIPT GOES HERE < /script >

(a) alert("Alert Text")

(b) Objects:
onclick="...Javascript to do...;return True|False"

(c) onMouseOver="...Javascript to do...;return True|False"

(d) onMouseOut="...Javascript to do...;return True|False"

(e) document.write("text to append to the document")

(f) window.open("URL", "window-name","window-options")

1.4 May

ASP.Net (Web) Programming in C# - Course Report (2014-05-27 11:56)

This course was last taught by me in 2009-10, if I recall correctly, for IInd M.Sc. (Maths) students in a deemed university in Andhra Pradesh, India.

Pre-requisites for the course: It so happened that students of this course had studied C and C++ languages. Students also had studied a database theory course and an SQL lab. course. So C++, relational database theory and SQL are pre-requisites for this course.

The first part of this course was/is a migration from C++ to C # which is available here.

Then ASP.NET Web programming was taught using the tutorial book, "Murach's ASP.NET 3.5 web programming with C # 2008" (referred to as Murach course book). [The current edition of this book seems to be Murach's ASP.NET 4.5 Web Programming with C # 2012. I guess that most of the course report given below would apply to the current edition too.] As the tutorial book has detailed instructions for students I had to prepare very little teaching material of my own. This document is a report on the course. The following are the topics that were covered in the course.

1) Introduction to Visual Studio 2008 and MSDN Library - Teaching notes

2) Chapter 1 of Murach course book, "An introduction to ASP.NET web programming". This covers quick exposure to user interface of pages of shopping cart ASP.NET application, hardware and software components of web apps, how static and dynamic web pages work, ASP.NET state handling, intro to ASP.NET application development, preview of .aspx and .cs source code & design mode view of shopping cart application, compilation of an ASP.NET application.

3) Chapter 2 of Murach course book, "How to develop a one-page web application". This covers creating a new web application (website) with Visual Studio 2008, sample Future Value program, using Design view to build a web form, flow layout, adding tables and server controls to a form, source and split views, aspx code for Future Value form, adding validation controls to a form, adding code to a form, using page and control events, testing the web application (Future Value app).

4) Chapter 3 of Murach course book, "How to develop a multi-page web application". This covers shopping cart application, design of order page and cart page, adding (business) classes Product and CartItem to the application, redirecting or transferring to another page, cross-page posting, coding absolute and relative URLs, creating and using Access data source, binding a drop-down list to a data source, using C # code to get data from a data source, using session state, aspx and C # code of the entire shopping cart application.

5) Chapter 4 of Murach course book, "How to test and debug an ASP.NET application". This covers creating a local IIS web site, creating a remote IIS web site, testing an ASP.NET application with default browser, testing an ASP.NET application with other than default browser, testing an application outside of Visual Studio, using the Exception Assistant, using the debugger, setting breakpoints, using tracepoints, working in break mode, controlling execution of the application (from the debugger), using Autos, Locals and Watch windows to monitor variables, using the Immediate window to work with values, using the Trace feature, creating custom Trace messages, writing information directly to HTTP output stream. Here are some teaching notes for this chapter.

6) For the HTML part of the course, due to time pressure, I specified Chapter 5, "A crash course in HTML" of the Murach course book as a reading assignment. Perhaps the thinking was that they would primarily use Design mode of ASP.NET and could fiddle with the HTML stuff on their own as they got more familiar with ASP.NET web programming. Please note that a suitable selection of topics from the MCLI HTML tutorial is available here: http://raviiyerteaches.wordpress.com/2014/04/12/html/ which was used for the Java Web Programming course (in 2005).

7) Chapter 6 & 7 of Murach course book, "How to work with server controls" & "How to use the validation controls". These cover types of server controls (Label, TextBox, Button, LinkButton, ImageButton, HyperLink, DropDownList, ListBox, CheckBox, CheckBoxList, RadioButton, RadioButtonList, Image, ImageMap, Calendar, Fileupload), button controls, handling control events (OnClick, OnCommand, OnSelectedIndexChanged, OnTextChanged ...), using e argument, using the Command event, text boxes, labels check boxes and radio buttons - aspx code, attributes, events and C # code, list controls - properties (Items, Rows, SelectedItem, SelectedIndex ...), list item object properties - Text, Value, Selected, List item collection member functions - Add, Insert, Remove, FindByValue, FindByText ..., hyperlink control, file upload control, image map control, calendar control, Validation controls (RequiredFieldValidator, CompareValidator, RangeValidator, RegularExpressionValidator, CustomValidator, ValidationSummary), Common validator properties (Display, ErrorMessage, Text ...), typical code for processing page that contains validation controls (Page.IsValid), using required field validator, using

compare validator, using range validator, using validation summary control, using validation groups, using a custom validator.

8) Chapter 8 of Murach course book, "How to manage state". This covers using view state, StateBag class, using session state, HttpSessionState class, when to save and retrieve session state items, using application state, HttpApplicationState class, working with application events, global.asax file, using cookies and URL encoding, HttpCookie class.

9) The database connectivity classes and ADO.NET concepts were also covered (if I recall correctly). Chapters 12 ("An introduction to database programming") & 13 ("How to use SQL data sources") cover it. The topics I would have or ideally should be covered are SqlConnection class, SqlCommand class, ProductList application that uses two SQL data sources, creating a SqlDataSource control, defining a connection (Configure Data Source dialog), saving connection string in web.config file, configuring the Select statement, creating a Where clause, how select parameters work, using DataList control, formatting a data list, data binding, how to bind a list control to a data source, Data Source Configuration wizard, Eval and Bind methods, aspx file for ProductList application, creating a data source that can update the database

10) Chapter 14 of Murach course book, "How to use the GridView Control". Some part of this chapter was covered if I recall correctly. Ideally these topics should be covered: GridView control, attributes, defining fields, sorting, paging, list application that uses GridView control, how to update GridView data, command fields, events raised by GridView control (RowDeleted, RowDeleting, RowEditing, RowUpdating, RowUpdated ...), insert a row in a GridView control, Category Maintenance application - aspx file and code-behind (.cs) file, template fields (EditItemTemplate),

11) A Hostel Store web project was given to students who were divided into teams of around four members each. I had given liberty to each team to define the functionality and user interface of their hostel store web app. Later they implemented their proposed functionality at least to some decent extent. The Student teams gave a talk-cum-demo at the end of the project work and at least some students were very happy that they now knew how to create small shopping centre type of applications using ASP.NET and SQLServer/Access database.

Migration from C++ to C# – Mini Course **(2014-05-30 08:53)**

This mini-course was taught by me last in 2009-10, if I recall correctly, for IInd M.Sc. (Maths) students in a deemed university in Andhra Pradesh, India, as part of a course on ASP.Net Web programming.

Pre-requisites for the course: It so happened that students of this course had studied C and C++ languages. So C++ is a pre-requisite for this course.

Migration from C++ to C #

[Note example C # source files referred in the Powerpoint presentations below as not (given) in the C # specification document are available here.]

1) Powerpoint presentation: CPPtoCsharp1. This covers: Course introduction, Course material for C++ to C # migration - Chapter 1, Introduction, of the document 'C # Language Specification Version 1.2' from Microsoft: http://download.microsoft.com/download/5/e/5/5e58be0a-b02b-41ac-a4a3-7a22286214ff/csharp %20language %20specification %20v1.2.doc (later versions of C # Language specification were available then itself but Chapter 1, Introduction, of this document was felt more suitable as a tutorial), C # features, Hello world app., Value and Reference types, comparison between Reference in C++ and C #, string class, boxing and unboxing, and program structure.

2) Powerpoint presentation: CPPtoCsharp2. This covers: List class - constructor, properties, read-only and read-write properties and indexer.

3) Powerpoint presentation: CPPtoCsharp3. This covers: event-driven programming, events, quick understanding of delegate, publisher-subscriber relationship, delegate type, event-delegate and event example.

4) Powerpoint presentation: CPPtoCsharp4. This covers: operators, destructors, struct, arrays, interface, IControl interface, class implements interface, interface knowledge reuse, interface - contract, interface variables/types, interface inheritance, interface - multiple inheritance, class - multiple interfaces, interface - methods, properties, events & indexers.

5) Powerpoint presentation: CPPtoCsharp5. This covers: enums, delegate, delegate vs. function pointers, delegate example, multi-cast delegate, event vs. delegate and attributes.

1.5 August

Advice to Fresh CS Graduates & PGs on Industry Jobs; Prototype vs. production programming **(2014-08-12 09:26)**

Here are two slides I may have last used in separate lectures/talks to M.Tech. (CS) students and/or M.Sc. (Maths) students around 2010 advising them on software industry jobs and on prototype programming quality vs. production programming quality. Please note that I modified the slides a bit before putting it up on this website/blog.

- Advice to Fresh Comp. Sc. Graduates & Post-Graduates On Industry Software Development Jobs; Prototype Quality Orientation To Production Quality Orientation (14 slides)

- M.Tech. research project programming vs. industry grade programming; Has Brian Kernighan's thoughts on prototype vs. production programming (9 slides)

Software development mini-project lab. courses - a report (2014-08-12 12:07)

To give M.Sc. (Maths) students who did fair amount of Computer Science/Information Technology theory and lab. courses as part of the M.Sc. (Maths) degree, exposure to software projects, I ran a few mini-project lab. courses over the years (in a deemed university in Andhra Pradesh, India). As the mini-projects themselves are the intellectual property of the educational institution, I am not in a position to share any details about them. However my first task during these mini projects was to expose students who had done programming courses and so knew programming, to the tasks involved in doing small software projects. After that was done I would assign four to five students to a mini-project of their choice but which was acceptable to me (I mean, I would see whether it was feasible given the time constraints and also whether it would meet the goals of the mini-project lab. course). The students would go through the entire cycle of putting down their imagined user requirements in a document, the design, planning for the coding work among the team members, the coding, unit testing, integration testing, some level of documentation of design and finally, a demonstration of the work with each team member demonstrating a part of the work. The evaluation took into account the individual team member's specific contribution to the software.

I was quite successful, IMHO, in using the coffee brewer design document of Jim Weirich which is available here: http://www.cs.unibo.it/ cianca/wwwpages/ ids/coffee.pdf to teach/expose the following concepts to students:

- Simple Use Cases to capture functionality
- Simple Object Oriented Analysis & Design which is captured in simple UML diagrams (Class diagrams mainly)

Additional materials I either used or recommended as optional reading were:

- Parlezuml Use Cases and Use Case Driven development: http://www.codemanship.co.uk/parlezuml/tutorials/usecases.htm
- Using Object-Oriented and UML Tools for Hardware Design: A Case Study: http://www.sie.arizona.edu/sysengr/publishedPapers/HVAC _UML.pdf [Though it is a hardware design case study I felt it was a good tutorial on OOAD and UML even for software design students.]

The mini-projects that the students did were typically database oriented projects with a web front-end using Java or ASP.Net. At least some students have come up to me after finishing the mini-project and expressed satisfaction and joy at having developed some not-insignificant software mini-project (as

against doing only programming assignments earlier) and that too as a team effort. So I think these mini-project courses were quite successful in achieving their objective of giving most students some exposure to software projects.

BlogBook v1.0,
LaTeX 2$_\varepsilon$ & GNU/Linux.
https://www.blogbooker.com

Edited: September 17, 2018